AlazarTech SDK
Programmer's Guide

Version 7.1.1

October 6, 2015

License Agreement

Important
By using this software you accept the following terms of this License Agreement. If you do not agree with these terms, you should not use the software and promptly return it for a refund.

Ownership
Alazar Technologies, Inc., retains the ownership of this copy of the enclosed software package. It is licensed to you for use under the following conditions:

Grant of License
You may only concurrently use the enclosed software on the computers that have an Alazar Technologies, Inc. waveform digitizer card plugged in (for example, if you have purchased one Alazar Technologies, Inc. card, you have a license for one concurrent usage). If the number of users of the software exceeds the number of Alazar Technologies, Inc. cards you have purchased, you must have a reasonable process in place to assure that the number of persons concurrently using the software does not exceed the number of Alazar Technologies, Inc. cards purchased.

You may transfer this software to another party if the other party agrees to the terms and conditions of the agreement and completes and returns a registration card to Alazar Technologies, Inc. The registration card is available by writing to Alazar Technologies, Inc. If you transfer the software, you must simultaneously transfer all documentation and related disks.

Restrictions
You may not copy the documentation or software except as described in the installation section of this manual. You may not distribute, rent, sub-lease or lease the software or documentation, including translating, decomposing, or disassembling, or creating derivative works. You may not reverse-engineer any part of this software, or produce any derivative work. You may not make telecommunication transmittal of this software.

Termination
This license and your right to use this software automatically terminates if you fail to comply with any provision of this license agreement.

Rights
Alazar Technologies, Inc. retains all rights not expressly granted. Nothing in this agreement constitutes a waiver of Alazar Technologies, Inc.'s rights under the Canadian and U.S. copyright laws or any other Federal or State law.

Limited Warranty
If you discover physical defects in the media, Alazar Technologies, Inc. will replace the media or documentation at no charge to you, provided you return the item to be replaced

with proof of payment to Alazar Technologies, Inc. during the 90-day period after having taken delivery of the software.

Alazar Technologies, Inc. excludes any and all implied warranties, including warranties of merchantability and fitness for a particular purpose and limits your remedy to return the software and documentation to Alazar Technologies, Inc. for replacement. Although Alazar Technologies, Inc. has tested the software and reviewed the documentation, ALAZAR TECHNOLOGIES, INC. MAKES NO WARRANTY OF REPRESENTATION, EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO THIS SOFTWARE OR DOCUMENTATION, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS SOFTWARE AND DOCUMENTATION IS LICENSED "as is" AND YOU, THE LICENSEE, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE. IN NO EVENT WILL ALAZAR TECHNOLOGIES, INC. BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THIS SOFTWARE OR DOCUMENTATION, even if advised of the possibility of such damages. In particular, Alazar Technologies, Inc. shall have no liability for any data acquired, stored or processed with this software, including the costs of recovering such data.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESSED OR IMPLIED. No Alazar Technologies, Inc. dealer, agent or employee is authorized to make any modifications or additions to this warranty.

Information in this document is subject to change without notice and does not represent a commitment on the part of Alazar Technologies, Inc. The software described in this document is furnished under this license agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy the software on any medium except as specifically allowed in the license agreement. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the written permission of Alazar Technologies, Inc.

Some jurisdictions do not allow the exclusion of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights, which vary from jurisdiction to jurisdiction.

Contents

1 Getting Started....1
1.1 Introduction....1
1.2 Programming environments....1
1.2.1 C/C++ Linux....1
1.2.2 C/C++ Windows....2
1.2.3 C#....2
1.2.4 LabVIEW....2
1.2.5 Python....3
1.2.6 MATLAB....3
1.2.7 VB.NET....3
1.2.8 C++/CLI....4
1.3 Sample code....4
1.4 Contacting us....5
2 Programmer's Guide....6
2.1 Addressing a board....6
2.1.1 Getting a board identifier....6
2.1.2 Getting a board handle....6
2.1.3 Closing a board handle....8
2.1.4 Using a board handle....8
2.2 Resetting a board....9
2.3 Configuring a board....9
2.3.1 Timebase....9
2.3.2 Input control....15
2.3.3 Trigger control....17
2.3.4 AUX I/O....23
2.3.5 Data Packing....26
2.3.6 Dual Edge Sampling....26
2.4 Acquiring data....27
2.4.1 Single port....27
2.4.2 Dual port AutoDMA....34
2.4.3 Buffer size and alignment....53
2.4.4 Data format....54
2.5 Processing data....54
2.5.1 Converting samples values to volts....54
2.5.2 Saving binary files....64
2.6 AlazarDSP....66
2.6.1 On-FPGA FFT Overview....66
2.6.2 General Programming Concepts....67
2.6.3 Program Flow....67
3 Advanced Topics....70
3.1 External Clock Issues for OCT applications....70
3.1.1 Diagnose External Clock Issues....70
3.1.2 K-clock deglitching firmware....71
3.2 AlazarSetTriggerOperationForScanning....72

4 Reference 76
4.1 Function Groups 76
4.1.1 Initialization 76
4.1.2 Status and information 76
4.1.3 Configuration and control 77
4.1.4 Acquisition 78
4.1.5 AlazarAbortAsyncRead 81
4.1.6 AlazarAbortAutoDma 83
4.1.7 AlazarAbortCapture 85
4.1.8 AlazarAllocBufferU16 86
4.1.9 AlazarAllocBufferU8 87
4.1.10 AlazarAsyncRead 88
4.1.11 AlazarBeforeAsyncRead 90
4.1.12 AlazarAutoCalibrate 96
4.1.13 AlazarBoardsFound 97
4.1.14 AlazarBoardsInSystemByHandle 98
4.1.15 AlazarBoardsInSystemBySystemID 99
4.1.16 AlazarBusy 100
4.1.17 AlazarClose 101
4.1.18 AlazarCloseAUTODma 102
4.1.19 AlazarConfigureAuxIO 103
4.1.20 AlazarConfigureLSB 106
4.1.21 AlazarConfigureRecordAverage 108
4.1.22 AlazarConfigureSampleSkipping 110
4.1.23 AlazarCoprocessorDownload 112
4.1.24 AlazarCoprocessorRegisterRead 114
4.1.25 AlazarCoprocessorRegisterWrite 115
4.1.26 AlazarCreateStreamFile 116
4.1.27 AlazarDSPAbortCapture 118
4.1.28 AlazarDSPGetBuffer 119
4.1.29 AlazarDSPGetModules 120
4.1.30 AlazarDSPGetNextBuffer 122
4.1.31 AlazarDSPGetInfo 123
4.1.32 AlazarDSPGenerateWindowFunction 125
4.1.33 AlazarErrorCodeToCluster 128
4.1.34 AlazarErrorToText 129
4.1.35 AlazarEvents 130
4.1.36 AlazarFFTSetup 131
4.1.37 AlazarFFTSetWindowFunction 133
4.1.38 AlazarFreeBufferU16 134
4.1.39 AlazarFreeBufferU8 135
4.1.40 AlazarFlushAutoDMA 136
4.1.41 AlazarForceTrigger 137
4.1.42 AlazarForceTriggerEnable 138
4.1.43 AlazarGetAutoDMAHeaderTimeStamp 139
4.1.44 AlazarGetAutoDMAHeaderValue 141

4.1.45 AlazarGetAutoDMAPtr 143
4.1.46 AlazarGetBoardBySystemHandle 145
4.1.47 AlazarGetBoardBySystemID 146
4.1.48 AlazarGetBoardKind 147
4.1.49 AlazarGetBoardRevision 149
4.1.50 AlazarGetChannelInfo 150
4.1.51 AlazarGetCPLDVersion 152
4.1.52 AlazarGetDriverVersion 153
4.1.53 AlazarGetMaxRecordsCapable 154
4.1.54 AlazarGetNextAutoDMABuffer 155
4.1.55 AlazarGetNextBuffer 158
4.1.56 AlazarGetParameter 159
4.1.57 AlazarGetParameterUL 162
4.1.58 AlazarGetSDKVersion 164
4.1.59 AlazarGetStatus 165
4.1.60 AlazarGetSystemHandle 166
4.1.61 AlazarGetTriggerAddress 167
4.1.62 AlazarGetTriggerTimestamp 169
4.1.63 AlazarGetWhoTriggeredBySystemHandle 171
4.1.64 AlazarGetWhoTriggeredBySystemID 173
4.1.65 AlazarHyperDisp 175
4.1.66 AlazarInputControl 178
4.1.67 AlazarInputControlEx 180
4.1.68 AlazarNumOfSystems 181
4.1.69 AlazarOEMDownLoadFPGA 182
4.1.70 AlazarOpen 183
4.1.71 AlazarParseFPGAName 184
4.1.72 AlazarPostAsyncBuffer 186
4.1.73 AlazarQueryCapability 188
4.1.74 AlazarRead 191
4.1.75 AlazarReadEx 193
4.1.76 AlazarResetTimeStamp 195
4.1.77 AlazarSamplesPerBuffer 196
4.1.78 AlazarSetBWLimit 197
4.1.79 AlazarSetCaptureClock 198
4.1.80 AlazarSetClockSwitchOver 200
4.1.81 AlazarSetExternalClockLevel 202
4.1.82 AlazarSetExternalTrigger 203
4.1.83 AlazarSetLED 205
4.1.84 AlazarSetParameter 206
4.1.85 AlazarSetParameterUL 208
4.1.86 AlazarSetRecordCount 210
4.1.87 AlazarSetRecordSize 212
4.1.88 AlazarSetTriggerDelay 214
4.1.89 AlazarSetTriggerOperation 216
4.1.90 AlazarSetTriggerOperationForScanning 219

4.1.91 AlazarSetTriggerTimeOut 221
4.1.92 AlazarSleepDevice 222
4.1.93 AlazarStartAutoDMA 223
4.1.94 AlazarStartCapture 226
4.1.95 AlazarStopAutoDMA 227
4.1.96 AlazarTriggered 228
4.1.97 AlazarWaitAsyncBufferComplete 229
4.1.98 AlazarWaitForBufferReady 231
4.1.99 AlazarWaitNextAsyncBufferComplete 233
4.2 Error Codes 235
4.3 Glossary 238
5 Appendix – Board Specific Information 239

1 Getting Started

1.1 Introduction

AlazarTech supplies device drivers for Windows and Linux that allow software to configure AlazarTech digitizers, and transfer sample data from the digitizer to application buffers.

The AlazarTech software developer's kit (ATS-SDK) includes header and library files required to call functions exported by these device drivers in user written applications, as well as documentation and sample code describing how to use these functions.

This document is a part of the ATS-SDK. It describes how to call functions exported by AlazarTech device drivers to control one or more digitizer boards. It is divided into the following sections:

- A programming guide that describes how to configure, and acquire data from, digitizer boards.
- A reference guide that describes the functions exported by the device drivers.

To get the most from your AlazarTech digitizer:

- Read the user manual supplied their digitizer board. It provides an overview of the digitizer hardware, as well as detailed specifications.
- Read the "Programmer's guide" section of this document. It describes how to program the digitizer hardware to make an acquisition, and to transfer sample data into application buffers.
- Browse the SDK sample programs. They include sample code that demonstrates how to make many types of acquisitions supported by the digitizer.

Note that this document includes descriptions of board specific features and options that may not be available on your digitizer board. Please refer your board's user manual for its specifications.

1.2 Programming environments

1.2.1 C/C++ Linux

C/C++ developers under Linux should include the following header files in source files that use functions exported by the ATS-SDK library.

```
#include "AlazarError.h"
#include "AlazarApi.h"
#include "AlazarCmd.h"
```

These modules should also link against libATSApi.so.

© 2003-2015 Alazar Technologies Inc. Version 7.1.1

The development package for Linux defaults to installing the header files in /usr/local/AlazarTech/include, and the library files in the standard library directory for the target distribution.

1.2.2 C/C++ Windows

C/C++ developers should include the following header files in source files that use functions exported by the API library.

```
#include "AlazarError.h"
#include "AlazarApi.h"
#include "AlazarCmd.h"
```

These applications should also link against the 32- or 64-bit version of ATSApi.lib, as required.

The SDK setup program installs the header files in "Samples_C\Include", and the library files in "Samples_C\Library".

1.2.3 C#

C# developers should either:

- Add the file AlazarApi.cs to their project; or
- Add a reference to AlazarApiNet.dll to their project.

The ATS-SDK includes a wrapper class that declares many of the constants and unmanaged functions exported by AlazarTech device drivers. This class is provided both as a C# source file (AlazarApi.cs), and as a compiled assembly (AlazarApiNet.dll).

The SDK setup program copies AlazarApi.cs to the "Samples_CSharp\AlazarApiNet\AlazarApiNet" directory and AlazarApiNet.dll to the "Samples_CSharp" directory.

Note that you can use the solution "Samples_CSharp\AlazarApiNet" to build AlazarApiNet.dll from AlazarApi.cs.

1.2.4 LabVIEW

LabVIEW developers can either:

- Use the sub-VIs provided with the ATS-SDK (recommended)
- Call functions from ATSApi.dll directly using the LabVIEW interface for shared libraries.

The ATS-SDK sub-VIs consists of a very thin wrapper on top of the functions exported by the ATS-SDK. The VIs are named after the functions that they wrap. They are located in "Samples_LabVIEW\Library", and are used by all the code samples available in "Samples_LabVIEW".

The only difference between the connector panes of the VIs and the C function signatures is that an error cluster is propagated through the VIs. If the input error cluster contains an error, the VI simply returns without doing anything.

The error cluster output depends on the function:

- If the function does not generate errors, the input error cluster is simply propagated to the output.
- If the function returns an error code, it is converted to a cluster and send to the output
- If the function can return errors using special return values, then these errors are detected by the VI, an appropriate error code is generated, converted to a cluster and sent to the output

1.2.5 Python

Python developers can use the atsapi.py module provided in the "Samples_Python\Library" directory. It provides a very thin wrapper around the AlazarTech C/C++ API, with only minor differences:

- The 'Alazar' prefixes have been removed from the function names, and the first letter is not capitalized. For example, 'AlazarAbortAsyncRead' becomes 'abortAsyncRead'.
- Board handles have been removed. Instead, a Board class has been added. All the functions that take a board handle as a parameter are moved to being member functions of the Board class.
- A DMABuffer convenience class has been added, that takes care of memory allocation of DMA transfers.

1.2.6 MATLAB

MATLAB developers can:

- Call functions exported by AlazarTech drivers DLL directly from MATLAB scripts and functions using the MATLAB 'calllib' function.
- Create a MEX-file dynamic link library to configure and acquire data from the digitizer, and call the mexFunction entry point of the DLL from MATLAB.

ATS-SDK samples demonstrate how to use the MATLAB "calllib" interface. They use prototype files to load the AlazarTech driver library into memory, and call AlazarDefs.m to define constants used by the AlazarTech library.

The ATS-SDK setup program installs AlazarDefs.m, alazarLoadLibrary.m, and other helper functions in the "Samples_MATLAB\Include" folder.

1.2.7 VB.NET

VB.NET programmers should include a reference to "Samples_CSharp\AlazarApiNet.dll" in their solutions. This assembly provides a .NET interface to the functions and constants defined in the ATS-SDK.

The ATS-SDK does not currently include VB.NET sample code. See the C# samples for .NET sample code.

1.2.8 C++/CLI

C++/CLI programmers should include a reference to "Samples_CSharp\AlazarApiNet.dll" in their solutions. This assembly provides a .NET interface to the functions and constants defined in the ATS-SDK.

The ATS-SDK does not currently include C++/CLI sample code. See the C# samples for .NET sample code.

1.3 Sample code

ATS-SDK includes sample programs that demonstrate how to configure and acquire data from AlazarTech digitizers.

The SDK setup program installs the sample programs to "C:\AlazarTech\ATS-SDK\%API_VERSION%" under Microsoft Windows, and "/usr/local/AlazarTech" under Linux. See the "ReadMe.htm" file in the ATS-SDK base directory for a description of the samples included.

Sample programs are available for the following programming environments in the following sub-directories:

Language	Sub-directory
C/C++	Samples_C
C#	Samples_CSharp
MATLAB	Samples_MATLAB
LabVIEW	Samples_LabVIEW
Python	Samples_Python

Note that the sample programs contain many parameters that should be modified. These lines of code are preceded by "TODO" comments. Please search for these lines and modify them as required for your application.

Many sample programs require a trigger input. These sample programs configure a board to trigger when a signal connected to its CH A rises through 0V. Before running these samples, connect a 1 kHz sine waveform of amplitude about 90% of the board's input range from a function generator to the CH A connector, or modify trigger parameters as required. For example, the ATS9360 has an input range of +/- 400 mV. For this board, a sine wave of 700 mVpp is appropriate. If an appropriate trigger signal is not supplied, these samples will fail with an acquisition timeout error.

1.4 Contacting us

Contact us if you have any questions or comments about this document, or the sample code.

Web	http://www.alazartech.com
Email	mailto:support@alazartech.com
Phone	+1-514-426-4899
Fax	+1-514-426-2723
Mail	Alazar Technologies Inc. 6600 Trans-Canada Highway, Suite 310 Pointe-Claire, QC Canada H9R 4S2

Note that you can download the latest drivers and documentation from our web site.
http://www.alazartech.com/support/downloads.htm

2 Programmer's Guide

2.1 Addressing a board

2.1.1 Getting a board identifier

AlazarTech organizes its digitizer boards into "board systems". A board system is a group of one or more digitizer boards that share trigger and clock signals. To create a "board system" comprised of two or more boards, the boards need to be connected together using an AlazarTech SyncBoard. All of the channels in a board system trigger and are sampled simultaneuously.

ATS-SDK assigns a "system identifier" number to each board system. The first system detected is assigned system ID number of 1.

ATS- SDK assigns a "board identifier" number to each board in a board system. This number uniquely identifies a board within its board system.

- If a digitizer board is not connected to any other boards using a SyncBoard, then the SDK assigns it a board ID of 1.
- If two or more boards are connected together using a SyncBoard, then the SDK assigns each board an ID number that depends on how the board is connected to the SycnBoard. The board connected to the "master" slot on the SyncBoard is the master board in the board system, and is assigned a board ID number of 1.

Call the AlazarNumOfSystems function to determine the number of board systems detected by the SDK, and call the AlazarBoardsInSystemBySystemID function to determine the number of boards in the board system specified by its system identifier.

The following code fragment lists the system and board identifiers of each board detected by the device drivers.

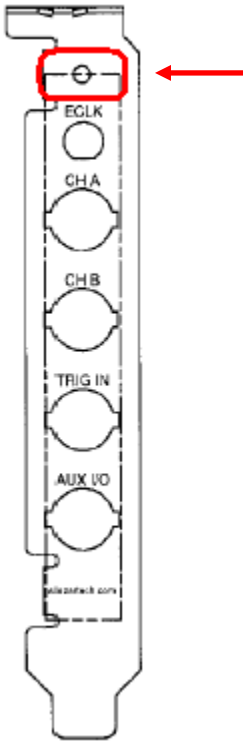
```
U32 systemCount = AlazarNumOfSystems();
for (U32 systemId = 1; systemId <= systemCount; systemId++)
{
   U32 boardCount = AlazarBoardsInSystemBySystemID(systemId);
   for (U32 boardId = 1; boardId <= boardCount; boardId++)
   {
      printf("Found SystemID %u Board ID = %u\n",
         systemId, boardId);
   }
}
```

2.1.2 Getting a board handle

ATS-SDK associates a handle with each digitizer board.

Most ATS-SDK functions require a board handle as a parameter. For example, the AlazarSetLED function allows an application to control the LED on the PCI/PCIe mounting bracket of a board specified by its handle.

Figure 2-1 PCI/PCIe mounting bracket LED



Use the AlazarGetBoardBySystemID API function to get a handle to a board specified by its system identifier and board identifier numbers.

2.1.2.1 Single board installations

If only one board is installed in a computer, ATS-SDK assigns it system ID 1 and board ID 1. The following code fragment gets a handle to such a board, and uses this handle to toggle the LED on the board's PCI/PCIe mounting bracket.

```
// Select a board
U32 systemId = 1;
U32 boardId = 1;

// Get a handle to the board
HANDLE boardHandle = AlazarGetBoardBySystemID(systemId, boardId);

// Toggle the LED on the board's PCI/PCIe mounting bracket
AlazarSetLED(boardHandle, LED_ON);
Sleep(500);
AlazarSetLED(boardHandle, LED_OFF);
```

2.1.2.2 Multiple board installations

If more than one board is installed in a PC, the boards are organized into board systems, and are assigned system and board identifier numbers. The following code fragment demonstrates how to obtain a handle to each board in such an installation, and use the handle to toggle the LED on the board's PCI/PCIe mounting bracket.

```
U32 systemCount = AlazarNumOfSystems();
for (U32 systemId = 1; systemId <= systemCount; systemId++)
{
   U32 boardCount = AlazarBoardsInSystemBySystemID(systemId);
   for (U32 boardId = 1; boardId <= boardCount; boardId++)
   {
      printf("SystemID %u Board ID = %u\n", systemId, boardId);

      // Get a handle to the board
      HANDLE handle = AlazarGetBoardBySystemID(systemId, boardId);

      // Toggle the LED on the board's PCI/PCIe mounting bracket
      AlazarSetLED(handle, LED_ON);
      Sleep(500);
      AlazarSetLED(handle, LED_OFF);
   }
}
```

2.1.2.3 System handles

Several ATS-SDK functions require a "system handle". A system handle is the handle of the master board in a board system.

- If a board is not connected to other boards using a SyncBoard, then its board handle is the system handle.
- If a board is connected to other boards using a SyncBoard, then the board that is connected to the master connector on the SyncBoard is the master board, and its board handle is the system handle.

2.1.3 Closing a board handle

ATS-SDK maintains a list of board handles in order to support master-slave board systems. The SDK creates board handles when it is loaded into memory, and destroys these handles when it is unloaded from memory. An application should not need to close a board handle.

2.1.4 Using a board handle

ATS-SDK includes a number of functions that return information about a board specified by its handle.

These functions include:

AlazarGetBoardKind	Get a board's model from its handle.
AlazarGetChannelInfo	Get the number of bits per sample, and on-board memory size in samples per channel.

AlazarGetCPLDVersion	Get the CPLD version of a board.
AlazarGetDriverVersion	Get the driver version of a board.
AlazarGetParameter	Get a board parameter as a signed 32-bit value.
AlazarGetParameterUL	Get a board parameter as an unsigned 32-bit value.
AlazarQueryCapability	Get a board capability as an unsigned 32-bit value.

The sample program "%ATS_SDK_DIR%\Samples\AlazarSysInfo" demonstrates how get a board handle, and use it to obtain board properties.

The API also exports functions that use a board handle to configure a board, arm it to make an acquisition, and transfer sample data from the board to application buffers. These topics are discussed in the following sections.

2.2 Resetting a board

The ATS-SDK resets all digitizer boards during its initialization procedure.

This initialization procedure automatically runs when the API library is loaded into memory.

- If an application statically links against the API library, the API resets all boards when the application is launched.
- If an application dynamically loads the API library, the API resets all boards when the application loads the API into memory.

Note that when an application using the API is launched, all digitizer boards are reset. If one application using the API is running when a second application using the API is launched, configuration settings written by the first application to a board may be lost. If a data transfer between the first application and a board was in progress, data corruption may occur.

2.3 Configuring a board

Before acquiring data from a board system, an application must configure the timebase, analog inputs, and trigger system settings of each board in the board system.

2.3.1 Timebase

The timebase of the ADC converters on AlazarTech digitizer boards may be supplied by:

- Its on-board oscillators.
- A user supplied external clock signal.
- An on-board PLL clocked by a user supplied 10 MHz reference signal.

2.3.1.1 Internal clock

To use on-board oscillators as a timebase, call AlazarSetCaptureClock specifying INTERNAL_CLOCK as the clock source identifier, and select the desired sample rate with a sample rate identifier appropriate for the board.

The following code fragment shows how to select a 10 MS/s internal sample rate.

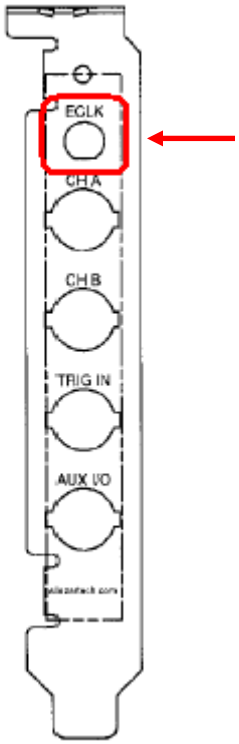
```
AlazarSetCaptureClock(
   handle,                 // HANDLE -- board handle
   INTERNAL_CLOCK,         // U32 -- clock source Id
   SAMPLE_RATE_10MSPS,     // U32 -- sample rate Id or value
   CLOCK_EDGE_RISING,      // U32 -- clock edge Id
   0                       // U32 -- decimation
   );
```

See AlazarSetCaptureClock or the board reference manual for a list of sample rate identifiers appropriate for a board.

2.3.1.2 External clock

AlazarTech boards optionally support using a user-supplied external clock signal input to the ECLK connector on its PCI/PCIe mounting bracket to clock its ADC converters.

Figure 2-2 External clock connector on PCI/PCIe mounting bracket.



To use an external clock signal as a timebase, call AlazarSetCaptureClock specifying SAMPLE_RATE_USER_DEF as the sample rate identifier, and select a clock source identifier appropriate for the board model and the external clock properties.

The following code fragment shows how to configure an ATS460 to acquire at 100 MS/s with a 100 MHz external clock.

```
AlazarSetCaptureClock(
```

```
    handle,                 // HANDLE -- board handle
    FAST_EXTERNAL_CLOCK,    // U32 -- clock source Id
    SAMPLE_RATE_USER_DEF,   // U32 -- sample rate Id or value
    CLOCK_EDGE_RISING,      // U32 -- clock edge Id
    0                       // U32 -- decimation
    );
```

See the board reference manual for the properties of an external clock signal that are appropriate for a board, and AlazarSetCaptureClock for a list of external clock source identifiers.

2.3.1.3 External clock level

Some boards allow adjusting the comparator level of the external clock input receiver to match the receiver to the clock signal supplied to the ECLK connector.

If necessary, call AlazarSetExternalClockLevel to set the relative external clock input receiver comparator level, in percent.

```
AlazarSetExternalClockLevel(
    handle,                 // HANDLE -- board handle
    level_pecent,           // float -- exernal clock level in percent
    );
```

2.3.1.4 10 MHz PLL

Some boards can generate a timebase from an on-board PLL clocked by user supplied external 10 MHz reference signal input to its ECLK connector.

2.3.1.4.1 ATS660

In 10 MHz PLL external clock mode, the ATS660 can generate a sample clock between 110 and 130 MHz, in 1 MHz, steps from an external 10 MHz reference input.

Call AlazarSetCaptureClock specifying EXTERNAL_CLOCK_10MHz_REF as the clock source identifier, the desired sample rate between 110 and 130 MHz in 1 MHz steps, and a decimation factor of 1 to 100000. Note that the decimation value should be one less than the desired decimation factor.

The following code fragment shows how to generate a 32.5 MS/s sample rate (130 MHz / 3) from a 10 MHz PLL external clock input.

```
AlazarSetCaptureClock(
    handle,                   // HANDLE -- board handle
    EXTERNAL_CLOCK_10MHz_REF, // U32 -- clock source Id
    130000000,                // U32 -- sample rate Id or value
    CLOCK_EDGE_RISING,        // U32 -- clock edge Id
    2                         // U32 -- decimation value
    );
```

2.3.1.4.2 ATS9325

In 10 MHz PLL external clock mode, the ATS9325 generates a 500 MHz sample clock from an external 10 MHz reference input. The 500 MS/s sample data can be decimated by a factor of 2, 4, or any multiple of 5.

Call AlazarSetCaptureClock specifying EXTERNAL_CLOCK_10MHz_REF as the clock source and 500 MHz as the sample rate, and select a decimation factor of 2, 4, or any multiple of 5 up to 100000.

For example, the following code fragment shows how to generate a 100 MS/s sample rate (500 MHz / 5) from a 10 MHz external clock input.

```
AlazarSetCaptureClock(
   handle,                    // HANDLE -- board handle
   EXTERNAL_CLOCK_10MHz_REF, // U32 -- clock source Id
   500000000,                 // U32 -- sample rate Id
   CLOCK_EDGE_RISING,         // U32 -- clock edge Id
   5                          // U32 -- decimation
   );
```

2.3.1.4.3 ATS9350 /ATS9351

In 10 MHz PLL external clock mode, the ATS9350 and ATS9351 generate a 500 MHz sample clock from an external 10 MHz reference input. The 500 MS/s sample data can be decimated by a factor of 1, 2, 4, or any multiple of 5.

Call AlazarSetCaptureClock specifying EXTERNAL_CLOCK_10MHz_REF as the clock source and 500 MHz as the sample rate, and select a decimation factor of 1, 2, 4, or any multiple of 5 up to 100000.

For example, the following code fragment shows how to generate a 100 MS/s sample rate (500 MHz / 5) from a 10 MHz external clock input.

```
AlazarSetCaptureClock(
   handle,                    // HANDLE -- board handle
   EXTERNAL_CLOCK_10MHz_REF, // U32 -- clock source Id
   500000000,                 // U32 -- sample rate Id
   CLOCK_EDGE_RISING,         // U32 -- clock edge Id
   5                          // U32 -- decimation
   );
```

2.3.1.4.4 ATS9360

In 10 MHz PLL external clock mode, the ATS9360 can generate any sample clock frequency between 300 MHz and 1800 MHz that is a multiple of 1 MHz.

Call AlazarSetCaptureClock specifying EXTERNAL_CLOCK_10MHz_REF as the clock source identifier, the desired sample rate between 300 MS/s and 1800 MS/s, and 1 as the decimation ratio. The sample rate must be a multiple of 1 MHz.

For example, the following code fragment shows how to generate a 1.382 GS/s sample clock from a 10 MHz reference.

```
AlazarSetCaptureClock(
   handle,                    // HANDLE -- board handle
   EXTERNAL_CLOCK_10MHz_REF,  // U32 -- clock source Id
   1382000000,                // U32 -- sample rate
   CLOCK_EDGE_RISING,         // U32 -- clock edge Id
   1                          // U32 -- decimation
   );
```

2.3.1.4.5 ATS9373

In 10 MHz PLL external clock mode, the ATS9373 can generate any sample clock frequency between 500 MHz and 2000 MHz that is a multiple of 1 MHz in either single or dual channel mode. In addition, it can generate any sample clock frequency between 2000 MHz and 4000 MHz that is a multiple of 2 MHz in single channel mode.

Call AlazarSetCaptureClock specifying EXTERNAL_CLOCK_10MHz_REF as the clock source identifier, the desired sample rate between 300 MS/s and 4000 MS/s, and 1 as the decimation ratio. The sample rate must be a multiple of 1 MHz in dual channel if the frequency is less than or equal to 2000 MHz, and a multiple of 2 MHz if the frequency is above 2000 MHz.

For example, the following code fragment shows how to generate a 1.382 GS/s sample clock from a 10 MHz reference.

```
AlazarSetCaptureClock(
   handle,                    // HANDLE -- board handle
   EXTERNAL_CLOCK_10MHz_REF,  // U32 -- clock source Id
   1382000000,                // U32 -- sample rate
   CLOCK_EDGE_RISING,         // U32 -- clock edge Id
   1                          // U32 -- decimation
   );
```

2.3.1.4.6 ATS9440

In 10 MHz PLL external clock mode, the ATS9440 can generate either a 125 MHz or 100 MHz sample clock from an external 10 MHz reference input. The 125 MS/s or 100 MS/s sample data can be decimated by a factor of 2, 4, or any multiple of 5.

Call AlazarSetCaptureClock specifying EXTERNAL_CLOCK_10MHz_REF as the clock source either 125 MHz or 100 MHz as the sample rate, and select a decimation radio between 1 and 100000.

For example, the following code fragment shows how to generate a 25 MS/s sample rate (125 MHz / 5) from a 10 MHz external clock input.

```
AlazarSetCaptureClock(
   handle,                    // HANDLE -- board handle
   EXTERNAL_CLOCK_10MHz_REF,  // U32 -- clock source Id
   125000000,                 // U32 -- sample rate Id
   CLOCK_EDGE_RISING,         // U32 -- clock edge Id
```

```
    5                           // U32 -- decimation
    );
```

2.3.1.4.7 ATS9462

In 10 MHz PLL external clock mode, the ATS9462 can generate a sample clock between 150 and 180 MHz in 1 MHz steps from an external 10 MHz reference input. Sample data can be decimated by a factor of 1 to 100000.

Call AlazarSetCaptureClock specifying EXTERNAL_CLOCK_10MHz_REF as the clock source, the desired sample rate between 150 and 180 MHz in 1 MHz steps, and the decimation factor of 1 to 100000. Note that the decimation value should be one less than the desired decimation factor.

For example, the following code fragment shows how to generate a 15 MS/s sample rate (150 MHz / 10) from a 10 MHz external clock input.

```
AlazarSetCaptureClock(
   handle,                  // HANDLE -- board handle
   EXTERNAL_CLOCK_10MHz_REF, // U32 -- clock source Id
   150000000,               // U32 -- sample rate Id or value
   CLOCK_EDGE_RISING,       // U32 -- clock edge Id
   9                        // U32 -- decimation value
   );
```

2.3.1.4.8 ATS9625/ATS9626

In 10 MHz PLL external clock mode, the ATS9625/ATS9626 can generate a 250 MHz sample clock from an external 10 MHz reference input. Sample data can be decimated by a factor of 1 to 100000.

Call AlazarSetCaptureClock specifying EXTERNAL_CLOCK_10MHz_REF as the clock source, 250 MHz has the sample rate value, and a decimation ratio of 1 to 100000.

For example, the following code fragment shows how to generate a 25 MS/s sample rate (250 MHz / 10) from a 10 MHz external clock input.

```
AlazarSetCaptureClock(
   handle,                  // HANDLE -- board handle
   EXTERNAL_CLOCK_10MHz_REF, // U32 -- clock source Id
   250000000,               // U32 -- sample rate Id or value
   CLOCK_EDGE_RISING,       // U32 -- clock edge Id
   10                       // U32 -- decimation value
   );
```

2.3.1.4.9 ATS9850

In 10 MHz PLL external clock mode, the ATS9850 generates a 500 MHz sample clock from an external 10 MHz reference input. The 500 MS/s sample data can be decimated by a factor of 1, 2, 4, or any multiple of 10.

Call AlazarSetCaptureClock specifying EXTERNAL_CLOCK_10MHz_REF as the clock source and 500 MHz as the sample rate value, and a decimation of 1, 2, 4, or any multiple of 10 up to 100000.

For example, the following code fragment shows how to generate a 125 MS/s sample rate (500 MHz / 4) from a 10 MHz external clock input.

```
AlazarSetCaptureClock(
   handle,                    // HANDLE -- board handle
   EXTERNAL_CLOCK_10MHz_REF, // U32 -- clock source Id
   500000000,                 // U32 -- sample rate Id or value
   CLOCK_EDGE_RISING,         // U32 -- clock edge Id
   4                          // U32 -- decimation value
   );
```

2.3.1.4.10 ATS9870

In 10 MHz PLL external clock mode, the ATS9870 generates a 1 GHz sample clock from an external 10 MHz reference input. The 1 GS/s sample data can be decimated by a factor of 1, 2, 4, or any multiple of 10.

Call AlazarSetCaptureClock specifying EXTERNAL_CLOCK_10MHz_REF as the clock source and 1 GHz as the sample rate value, and a decimation of 1, 2, 4, or any multiple of 10 up to 100000.

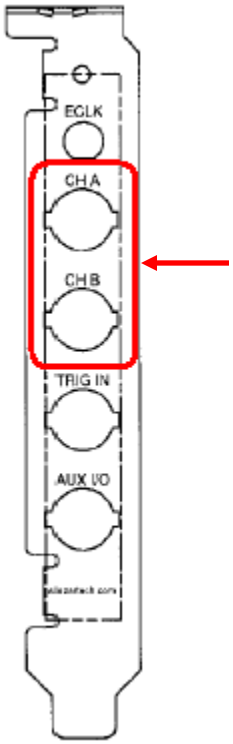
For example, the following code fragment shows how to generate a 250 MS/s sample rate (1 GHz / 4) from a 10 MHz external clock input.

```
AlazarSetCaptureClock(
   handle,                    // HANDLE -- board handle
   EXTERNAL_CLOCK_10MHz_REF, // U32 -- clock source Id
   1000000000,                // U32 -- sample rate Id or value
   CLOCK_EDGE_RISING,         // U32 -- clock edge Id
   4                          // U32 -- decimation value
   );
```

2.3.2 Input control

AlazarTech digitizers have analog amplifier sections that process the signals input to its analog input connectors before they are sampled by its ADC converters. The gain, coupling, and termination of the amplifier sections should be configured to match the properties of the input signals.

Figure 2-3 CHA and CHB connectors on a PCI/PCIe mounting bracket.



2.3.2.1 Input range, coupling, and impedance

Call AlazarInputControl to specify the desired input range, termination, and coupling of an input channel.

The following code fragment configures input CH A for a range of ± 800 mV, DC coupling, and 50Ω termination.

```
AlazarInputControl(
    boardHandle,            // HANDLE -- board handle
    CHANNEL_A,              // U8 -- input channel
    DC_COUPLING,            // U32 -- input coupling id
    INPUT_RANGE_PM_800_MV,  // U32 -- input range id
    IMPEDANCE_50_OHM        // U32 -- input impedance id
    );
```

See AlazarInputControl and the board reference manual for a list of input range, coupling, and impedance identifiers appropriate for the board.

2.3.2.2 Bandwidth filter

Some digitizers have a low pass filters that attenuate signals above about 20 MHz. By default, these filters are disabled. Call AlazarSetBWLimit to enable or disable the bandwidth limit filter.

The following code fragment enables the CH A bandwidth limit filter.

```
AlazarSetBWLimit (
  boardHandle,            // HANDLE -- board handle
  CHANNEL_A,              // U32 -- channel identifier
  1                       // U32 -- 0 = disable, 1 = enable
  );
```

2.3.2.3 Amplifier bypass

Some digitizer models support "amplifier bypass" mode. In this mode, the analog signal supplied to an input connector is connected directly the ADC driver of that channel, bypassing its amplifier section.

Amplifier bypass mode must be enabled in hardware either through DIP switches on the board, or as a factory option. Once enabled in hardware, the following code fragment shows how to configure this option in software.

```
AlazarInputControl(
    handle,                 // HANDLE -- board handle
    CHANNEL_A,              // U8 -- input channel
    DC_COUPLING,            // U32 -- not used
    INPUT_RANGE_HI_FI,      // U32 -- input range id
    IMPEDANCE_50_OHM        // U32 -- not used
    );
```

Note that when amplifier bypass mode option is enabled for an input channel, the channel's full-scale input range is fixed. The following table lists the nominal full-scale input range values that may be used to convert sample code values to volts.

Model	Full scale input range
ATS460	± 525 mV
ATS660	± 550 mV
ATS9325/ATS9350	± 200 mV
ATS9351	± 400 mV
ATS9462	± 550 mV
ATS9850/ATS9870	± 256 mV

See your board's hardware reference manual for more information about using amplifier bypass.

2.3.3 Trigger control

AlazarTech digitizer boards have a flexible triggering system with two separate trigger engines that can be used independently, or combined together to generate trigger events.

2.3.3.1 AlazarSetTriggerOperation

Use the AlazarSetTriggerOperation API function to configure each of the two trigger engines, and to specify how they should be used to generate trigger events.

```
RETURN_CODE
AlazarSetTriggerOperation (
   HANDLE handle,
   U32 TriggerOperation,
   U32 TriggerEngineId1,
   U32 SourceId1,
   U32 SlopeId1,
   U32 Level1,
   U32 TriggerEngineId2,
   U32 SourceId2,
   U32 SlopeId2,
   U32 Level2
   );
```

The following paragraphs describe each of the function's parameters, and provide examples showing how to use the function.

2.3.3.1.1 Trigger engine

The trigger engine identifier parameter specifies which of the two trigger engines you wish to configure. The parameter may have one of the following values.

Identifier	Value	Description
TRIG_ENGINE_J	0	Configure trigger engine J
TRIG_ENGINE_K	1	Configure trigger engine K

2.3.3.1.2 Data source

The data source identifier parameter selects the where the specified trigger engine should get its data. Refer to the documentation of the AlazarSetTriggerOperation function for a list of all possible values.

2.3.3.1.3 Trigger slope

The trigger slope identifier parameter selects if the output of the specified trigger engine should become active when sample values from the specified trigger source rise above, or fall below, a specified level. The parameter may have one of the following values.

Identifier	Value	Description
TRIGGER_SLOPE_POSITIVE	1	The trigger engine output goes from low to high when sample values from the trigger source rise above a specified level.
TRIGGER_SLOPE_NEGATIVE	2	The trigger engine output goes from low to high when sample values from the trigger source fall below a specified level.

2.3.3.1.4 Trigger level

The trigger level parameter sets the level that the trigger source must rise above, or fall below, for the selected trigger engine to become active. The trigger level is specified as an unsigned 8-bit code that represents a fraction of the full scale input range of the trigger source; 0 represents the negative full-scale input, 128 represents a 0 volt input, and 255 represents the positive full-scale input.

For example, if the trigger source is CH A, and the CH A input range is ± 800 mV, then 0 represents a –800 mV trigger level, 128 represents a 0 V trigger level, and 255 represents +800 mV trigger level.

In general, the trigger level value is given by:

```
TriggerLevelCode = 128 + 127 * TriggerLevelVolts / InputRangeVolts.
```

The following table gives examples of how trigger level codes map to trigger levels in volts according to the full-scale input range of the trigger source.

Trigger level code	Trigger level as fraction of source input range	Trigger level if source has ±1 V input range	Trigger level if source has ±5 V input range
0	-100%	-1V	-5V
64	-50%	-500 mV	-2.5 V
96	-25%	-250 mV	-1.25 V
128	0%	0 V	0 V
160	+25 %	250 mV	1.25 V
192	+50%	+500 mV	+2.5 V
255	+100%	+1V	+5V

2.3.3.1.5 Trigger operation

Finally, the trigger operation identifier specifies how the outputs of from the two trigger engines are combined to generate trigger events. This parameter may have one of the following values where the symbol T_j represents the output of trigger engine J, and T_k represents the output of trigger engine K.

Identifier	Value	Meaning Generate a trigger event when…
TRIG_ENGINE_OP_J	0	T_j goes low to high.
TRIG_ENGINE_OP_K	1	T_k goes low to high.
TRIG_ENGINE_OP_J_OR_K	2	T_j goes low to high, or T_k goes low to high.
TRIG_ENGINE_OP_J_AND_K	3	(T_j AND T_k) goes low to high.
TRIG_ENGINE_OP_J_XOR_K	4	(T_j XOR T_k) goes low to high.
TRIG_ENGINE_OP_J_AND_NOT_K	5	(T_j AND (NOT T_k)) goes low to high.
TRIG_ENGINE_OP_NOT_J_AND_K	6	((NOT T_j)AND T_k) goes low to high.

2.3.3.2 AlazarSetTriggerOperation examples

The following code fragment configures a board to trigger when the signal connected to CH A rises above 0V. This example only uses trigger engine J.

```
AlazarSetTriggerOperation(
   handle,                 // HANDLE -- board handle
   TRIG_ENGINE_OP_J,       // U32 -- trigger operation
   TRIG_ENGINE_J,          // U32 -- trigger engine id
   TRIG_CHAN_A,            // U32 -- trigger source id
   TRIGGER_SLOPE_POSITIVE, // U32 -- trigger slope id
   128,                    // U32 -- trigger level (128 = 0V)
   TRIG_ENGINE_K,          // U32 -- trigger engine id
   TRIG_DISABLE,           // U32 -- trigger source id for engine K
   TRIGGER_SLOPE_POSITIVE, // U32 -- trigger slope id
   128                     // U32 -- trigger level (0 – 255)
);
```

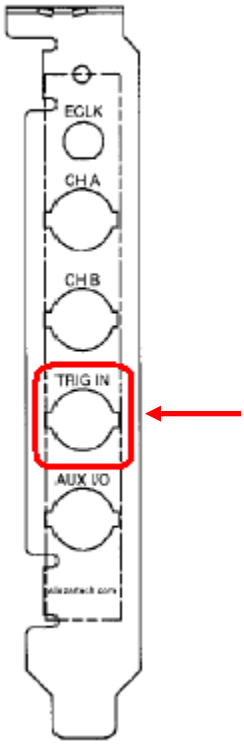
The following code fragment configures a board to trigger when the signal connected to CH B rises above 500 mV, or falls below -200 mV, if CH B's input range is ±1V. This example uses both trigger engine J and K.

```
double inputRange_volts = 1.; // ±1V range

double TriggerLevelJ_volts = .5;  // +500 mV trigger level
U32 triggerLevelJ =               // U32 -- trigger level J (192)
   (U32)(128 + 127 * triggerLevelJ_volts / inputRange_volts);

double triggerLevelK_volts = -.2; // -200 mV trigger level
U32 triggerLevelK =               // U32 – trigger level K (103)
   (U32)(128 + 127 * triggerLevelK_volts / inputRange_volts);

AlazarSetTriggerOperation(
   handle,                 // HANDLE -- board handle
   TRIG_ENGINE_OP_J_OR_K,  // U32 -- trigger operation
   TRIG_ENGINE_J,          // U32 -- trigger engine id
   TRIG_CHAN_B,            // U32 -- trigger source id
   TRIGGER_SLOPE_POSITIVE, // U32 -- trigger slope id
   triggerLevelJ,          // U32 -- trigger level from 0 to 255
   TRIG_ENGINE_K,          // U32 -- trigger engine id
   TRIG_DISABLE,           // U32 -- trigger source id for engine K
   TRIGGER_SLOPE_POSITIVE, // U32 -- trigger slope id
   triggerLevelK,          // U32 -- trigger level from 0 to 255
);
```

2.3.3.3 External trigger

AlazarTech digitizer boards can trigger on a signal connected to its TRIG IN connector.

Figure 2-4 External trigger connector on PCI/PCIe mounting bracket.



To use an external trigger input:

- Call AlazarSetTriggerOperation with TRIG_EXTERNAL as the trigger source identifier of at least one of the trigger engines; and
- Call AlazarSetExternalTrigger to select the range and coupling of the external trigger input.

The following code fragment configures a board to trigger when the signal connected to the TRIG IN falls below +2 V, assuming the signal's range is less than ± 5V with DC coupling.

```
// Calculate the trigger level code from the level and range

double triggerLevel_volts = 2.; // trigger level
double triggerRange_volts = 5.; // input range
U32 triggerLevel_code =
   (U32)(128 + 127 * triggerLevel_volts / triggerRange_volts);

// Configure trigger engine J to generate a trigger event
// on the falling edge of an external trigger signal.

AlazarSetTriggerOperation(
   handle,                 // HANDLE -- board handle
   TRIG_ENGINE_OP_J,       // U32 -- trigger operation
   TRIG_ENGINE_J,          // U32 -- trigger engine id
   TRIG_EXTERNAL,          // U32 -- trigger source id
   TRIGGER_SLOPE_NEGATIVE, // U32 -- trigger slope id
```

```
    triggerLevel,           // U32 -- trigger level (0 - 255)
    TRIG_ENGINE_K,          // U32 -- trigger engine id
    TRIG_DISABLE,           // U32 -- trigger source id for engine K
    TRIGGER_SLOPE_POSITIVE, // U32 -- trigger slope id
    128                     // U32 -- trigger level (0 - 255)
);

// Configure the external trigger input to +/-5V range,
// with DC coupling

AlazarSetExternalTrigger(
    handle,                  // HANDLE -- board handle
    DC_COUPLING,             // U32 -- coupling id
    ETR_5V                   // U32 -- external range id
);
```

2.3.3.4 Trigger timeout

AlazarTech digitizer boards can be configured to automatically trigger when the board is waiting for a trigger event, but no trigger events arrive after a specified time interval. This behavior is similar to the "automatic" trigger mode of oscilloscopes, and may be useful to capture waveforms when trigger conditions are unknown.

Call AlazarSetTriggerTimeOut to specify the amount of time that a board should wait for a hardware trigger event before automatically generating a software trigger event and, as a result, acquiring one record. The timeout value is expressed in 10 µs units, where 0 means disable the timeout counter and wait forever for a trigger event.

The trigger timeout value should be set to zero once stable trigger parameters have been found. Otherwise, a board may generate unexpected trigger events if the trigger timeout interval expires before a hardware trigger event occurs.

The following code fragment configures a board to automatically trigger and acquire one record if it does not receive a trigger event after 1 ms.

```
double timeout_sec = 1.e-3;    // 1 ms
U32 timeout_ticks = (U32)(timeout_sec / 10.e-6 + 0.5);

AlazarSetTriggerTimeOut(
    boardHandle,      // HANDLE -- board handle
    timeout_ticks     // U32 - timeout_sec / 10.e-6 (0 = infinite)
    );
```

The following code fragment configures a board to wait forever for trigger events.

```
AlazarSetTriggerTimeOut(
    boardHandle,  // HANDLE -- board handle
    0             // U32 -- timeout_sec / 10.e-6 (0 = infinite)
    );
```

2.3.3.5 Trigger delay

An AlazarTech digitizer board can be configured to wait for a specified amount of time after it receives a trigger event before capturing a record for the trigger.

Call AlazarSetTriggerDelay to specify a time, in sample clock periods, to wait after receiving a trigger event for a record before capturing samples for that record.

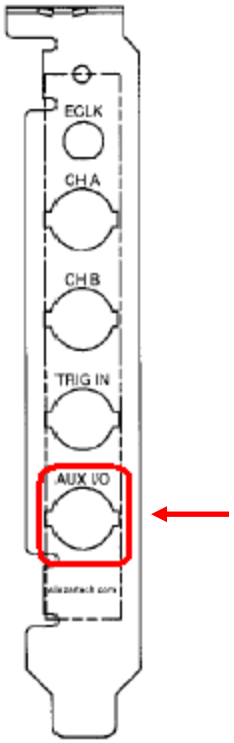
The following code fragment shows how to set a trigger delay of 1 ms, given a sample rate of 100 MS/s.

```
double triggerDelay_sec = 1.e-3;    // 1 ms
double samplesPerSec = 100.e6;      // 100 MS/s
U32 triggerDelay_samples =
   (U32)(triggerDelay_sec * samplesPerSec + 0.5);

AlazarSetTriggerDelay(
   boardHandle,              // HANDLE -- board handle
   triggerDelay_samples      // U32 -- trigger delay in samples
   );
```

2.3.4 AUX I/O

AlazarTech digitizer boards with an AUX I/O connector can be configured to supply a 5V TTL-level output signal, or to receive a TTL-level input signal on this connector.

Figure 2-5 Auxiliary I/O connector on PCI/PCIe mounting bracket



Use AlazarConfigureAuxIO to configure the function of the AUX I/O connector.

The ATS9440 has two AUX I/O connectors: AUX I/O 1 and AUX I/O 2. AUX I/O 1 is configured by firmware as a trigger output signal, while AUX I/O 2 is configured by software using AlazarConfigureAuxIO. A custom FPGA is required to change the operation of AUX I/O 1.

The ATS9625 and ATS9626 have two AUX I/O connectors: AUX1 and AUX2. AUX1 is configured by by software using AlazarConfigureAuxIO, while AUX2 is configured by the main FPGA as a trigger output signal by default. AUX2 can be controlled by its user-programmable FPGA as desired by the FPGA designer.

2.3.4.1 Trigger output

The AUX I/O connector can be configured to supply a trigger output signal, where the edge of the trigger output signal is synchronized with the edge of the sample clock. Note that this is the default power-on mode for the AUX I/O connector.

The following code fragment configures the AUX I/O connector as a trigger output signal.

```
AlazarConfigureAuxIO(
    handle,                 // HANDLE -- board handle
    AUX_OUT_TRIGGER,        // U32 -- mode
    0                       // U32 -- parameter
    );
```

2.3.4.2 Pacer output

The AUX I/O connector can be configured to output the sample clock divided by a programmable value. This option may be used to generate a clock signal synchronized with the sample clock of the digitizer board.

The following code fragment generates a 10 MHz signal on an AUX I/O connector, given a sample rate of 180 MS/s.

```
AlazarConfigureAuxIO(
    handle,                 // HANDLE -- board handle
    AUX_OUT_PACER,          // U32 -- mode
    18                      // U32 -- sample clock divider
    );
```

Note that the sample rate divider value must be greater than 2, and that the signal output may be limited by the bandwidth of the output's TTL drivers.

2.3.4.3 Digital output

The AUX I/O connector can be configured to output a TTL high or low signal. This mode allows a programmer to use the AUX I/O connector as a general purpose digital output.

The following code fragment configures the AUX I/O connector as a digital output.

```
AlazarConfigureAuxIO(
    handle,                 // HANDLE -- board handle
    AUX_OUT_SERIAL_DATA,    // U32 -- mode
    0                       // U32 -- 0 = low, 1 = high
    );
```

2.3.4.4 Trigger enable input

The AUX I/O connector can be configured as an AutoDMA trigger enable input signal. When enabled, a board will:

- Wait for a rising or falling edge on the AUX I/O.
- Wait for the number of trigger events necessary to capture the number of "records per buffer" in one AutoDMA segment specified at the start of the acquisition.
- Repeat.

The following code fragment configures the AUX I/O connector to acquire "records per buffer" records after it receives the rising edge of a TTL pulse connected on the AUX I/O connector.

```
AlazarConfigureAuxIO(
    handle,                 // HANDLE -- board handle
    AUX_IN_TRIGGER_ENABLE,  // U32 -- mode
    TRIGGER_SLOPE_POSITIVE  // U32 -- parameter
    );
```

See section 2.4.2.8 "AutoDMA Scanning Applications" for more information.

2.3.4.5 Digital input

The AUX I/O connector can be configured to read the TTL level of a signal input to the AUX connector. This mode allows a programmer to use the AUX I/O connector as a general purpose digital input.

The following code fragment configures the AUX I/O connector as a digital input.

```
AlazarConfigureAuxIO(
    handle,                 // HANDLE -- board handle
    AUX_IN_AUXILIARY,       // U32 -- mode
    0                       // U32 -- not used
    );
```

Once configured as a serial input, the following code fragment reads the AUX input level.

```
long level;
AlazarGetParameter(
    handle,                 // HANDLE -- board handle
    0,                      // U8 -- channel
```

```
        GET_AUX_INPUT_LEVEL,    // U32 -- parameter
        &level                  // long* -- 0 = low, 1 = high
        );
```

2.3.5 Data Packing

By default, all the boards that have more than 8-bit per sample sampling transfer data to the host computer with 2 bytes (16 bit) per sample. This behavior can be changed on some boards by packing the data, either to 8- or 12-bits per sample. This is done by calling the AlazarSetParameter function with the PACK_MODE parameter and a packing option (either PACK_DEFAULT, PACK_8_BITS_PER_SAMPLE or PACK_12_BITS_PER_SAMPLE). The parameter must be set before calling AlazarBeforeAsyncRead.

For a list of boards that implement 8-bit packing, 12-bit packing and both; please refer to *Table 9 – Miscellaneous Features Support.*

2.3.6 Dual Edge Sampling

Some AlazarTech digitizers are capable of dual edge sampling (DES), meaning that sample data is acquired both at the rising and falling edge of the clock signal. This mode can apply both to internal and external clocks. For example, ATS9373 is capable of 2 GS/s sampling in **non**-DES mode, and 4 GS/s in DES mode.

When using the internal clock, DES sampling is activated automatically. Data must be acquired from channel A only. To use DES sampling in external clock mode, one must call AlazarSetParameter as follows **before** configuring the board:

```
AlazarSetParameterUL(
        handle,                 // HANDLE -- board handle
        channelMask,            // U8 -- channel to acquire
        SET_ADC_MODE,
        ADC_MODE_DES
        );
```

Programs that wish to use DES-capable digitizers in non-DES mode (i.e. ATS9373 at sampling frequencies at or below 2GS/s) do not need to be modified.

2.4 Acquiring data

AlazarTech digitizers may be configured to acquire in one of the following modes:

- "Single port" mode acquires data to on-board memory and then, after the acquisition is complete, transfers data from on-board memory to application buffers.
- "Dual port AutoDMA" mode acquires to on-board memory while, at the same time, transferring data from on-board memory to application buffers.

2.4.1 Single port

The single-port acquisition API allows an application to capture records to on-board memory – one per trigger event – and transfer records from on-board to host memory. Data acquisition and data transfer are made serially, so trigger events may be missed if they occur during data transfers.

The single port acquisition API may be used if:

- A board has single-port or dual-port on-board memory.
- An application can miss trigger events that occur while it is transferring data from on-board to host memory.

The singe port acquisition API must be used if:

- A board does not have dual-port or FIFO on-board memory.
- An application acquires data at an average rate that is greater than maximum transfer rate of the board's PCI or PCIe host bus interface.

Ultrasonic testing, OCT, radar, imaging and similar applications should not use the single-port acquisition API; rather, they should use the dual-port acquisition API described in section 2.4.2 below.

2.4.1.1 Acquiring to on-board memory

2.4.1.1.1 All channels mode

By default, AlazarTech digitizer boards share on-board memory equally between both of a board's input channels. A single-port acquisition in dual-channel mode captures samples from both input channels simultaneously to on-board memory and, after the acquisition is complete, allows samples from either input channel to be transferred from on-board memory to an application buffer.

To program a board acquire to on-board memory in dual-channel mode:

1. Call AlazarSetRecordSize to set the number of samples per record, where a record may contain samples before and after its trigger event.
2. Call AlazarSetRecordCount to set the number records per acquisition – the board captures one record per trigger event.
3. Call AlazarStartCapture to arm the board to wait for trigger events.

4. Call AlazarBusy in a loop to poll until the board has received all trigger events in the acquisition, and has captured all records to on-board memory.
5. Call AlazarRead, AlazarReadEx, or AlazarHyperDisp to transfer records from on-board memory to host memory.
6. Repeat from step 3, if necessary.

The following code fragment acquires to on board memory with on-board memory shared between both input channels.

```
// 1. Set record size

AlazarSetRecordSize (
    boardHandle,            // HANDLE -- board handle
    preTriggerSamples,      // U32 -- pre-trigger samples
    postTriggerSamples      // U32 -- post-trigger samples
    );

// 2. Set record count

AlazarSetRecordCount(
    boardHandle,            // HANDLE -- board handle
    recordsPerCapture       // U32 -- records per acquisition
    );

// 3. Arm the board to wait for trigger events

AlazarStartCapture(boardHandle);

// 4. Wait for the board to receive all trigger events
// and capture all records to on-board memory

while (AlazarBusy (boardHandle))
{
    // The acquisition is in progress
}

// 5. The acquisition is complete. Call AlazarRead or
// AlazarHyperDisp to transfer records from on-board memory
// to your buffer.
```

2.4.1.1.2 Single channel mode

ATS9325, ATS9350, ATS9351, ATS9440, ATS9625, ATS9626, ATS9850, and ATS9870 and digitizer boards can be configured to dedicate all on-board memory to one of a board's input channels. A single-port acquisition in single-channel mode only captures samples from the specified channel to on-board memory and, after the acquisition is complete, only allows samples from the specified channel to be transferred from on-board memory to an application buffer.

To program a board acquire to on-board memory in single-channel mode:

1. Call AlazarSetRecordSize to set the number of samples per record, where a record may contain samples before and after its trigger event.

2. Call AlazarSetRecordCount to set the number records per acquisition – the board captures one record per trigger event.
3. Call AlazarSetParameter with the parameter SET_SINGLE_CHANNEL_MODE, and specify the channel to use all memory.
4. Call AlazarStartCapture to arm the board to wait for trigger events.
5. Call AlazarBusy in a loop to poll until the board has received all trigger events in the acquisition, and has captured all records to on-board memory.
6. Call AlazarRead, AlazarReadEx, or AlazarHyperDisp to transfer records from on-board memory to host memory.
7. Repeat from step 3, if necessary.

The following code fragment acquires to on-board memory from CH A in single channel mode.

```
// 1. Set record size

AlazarSetRecordSize (
    boardHandle,            // HANDLE -- board handle
    preTriggerSamples,      // U32 -- pre-trigger samples
    postTriggerSamples      // U32 -- post-trigger samples
    );

// 2. Set record count

AlazarSetRecordCount(
    boardHandle,            // HANDLE -- board handle
    recordsPerCapture       // U32 -- records per acquisition
    );

// 3. Enable single channel mode

AlazarSetParameter(
    boardHandle,             // HANDLE -- board handle
    0,                       // U8 -- channel Id (not used)
    SET_SINGLE_CHANNEL_MODE, // U32 -- parameter
    CHANNEL_A                // long - CHANNEL_A or CHANNEL_B
    );

// 4. Arm the board to wait for trigger events

AlazarStartCapture(boardHandle);

// 5. Wait for the board to receive all trigger events
// and capture all records to on-board memory

while (AlazarBusy (boardHandle))
{
    // The acquisition is in progress
}

// 6. The acquisition is complete. Call AlazarRead or
// AlazarHyperDisp to transfer records from on-board memory
// to your buffer.
```

Note that a call to AlazarSetParameter must be made before each call to AlazarStartCapture.

If the of number of samples per record specified in AlazarSetRecordSize is greater than the maximum number of samples per channel in dual-channel mode, but is less than the maximum number of samples per record in single-channel mode, and AlazarSetParameter is not called before calling AlazarStartCapture, then AlazarStartCapture will fail with error ApiNotSupportedInDualChannelMode (591).

2.4.1.2 Using AlazarRead

Use AlazarRead to transfer samples from records acquired to on-board memory to a buffer in host memory.

2.4.1.2.1 Transferring full records

The following code fragment transfers a full CH A record from on-board memory to a buffer in host memory.

```
// Allocate a buffer to hold one record.
// Note that the buffer must be at least 16 samples
// larger than the number of samples per record.

U32 allocBytes = bytesPerSample * (samplesPerRecord + 16);
void* buffer = malloc(allocBytes);

// Transfer a CHA record into our buffer
AlazarRead (
    boardHandle,                // HANDLE -- board handle
    CHANNEL_A,                  // U32 -- channel Id
    buffer,                     // void* -- buffer
    bytesPerSample,             // int -- bytes per sample
    (long) record,              // long -- record (1 indexed)
    -((long)preTriggerSamples), // long -- trigger offset
    samplesPerRecord            // U32 -- samples to transfer
    );
```

See "%ATS_SDK_DIR%\Samples\SinglePort\AR" for a complete sample program that demonstrates how to use AlazarRead to read full records.

2.4.1.2.2 Transferring partial records

AlazarRead can transfer a segment of a record from on-board memory to a buffer in host memory. This may be useful if:

- The number of bytes in a full record in on-board memory exceeds the buffer size in bytes that an application can allocate in host memory.
- An application wishes to reduce the time required for data transfer when it acquires relatively long records to on-board memory, but is only interested in a relatively small part of the record.

Use the "TransferOffset" parameter in the call to AlazarRead to specify the offset, in samples from the trigger position in the record, of the first sample to transfer from on-board memory to the application buffer. And use the "TransferLength" parameter to specify the number of samples to transfer from on-board memory to the application buffer, where this number of samples may be less than the number of samples per record.

The following code fragment divides a record into segments, and transfers the segments from on-board to host memory.

```
// Allocate a buffer to hold one record segment.
// Note that the buffer must be at least 16 samples
// larger than the number of samples per buffer.

U32 allocBytes = bytesPerSample * (samplesPerBuffer + 16);
void* buffer = malloc(allocBytes);

// Transfer a record in segments from on-board memory

U32 samplesToRead = samplesPerRecord;
long triggerOffset_samples = -(long)preTriggerSamples;

while (samplesToRead > 0)
{
    // Transfer a record segment from on-board memory

    U32 samplesThisRead;
    if (samplesToRead > samplesPerBuffer)
        samplesThisRead = samplesPerBuffer;
    else
        samplesThisRead = samplesToRead;

    AlazarRead (
        boardHandle,                // HANDLE -- board handle
        CHANNEL_A,                  // U32 -- channel Id
        buffer,                     // void* -- buffer
        bytesPerSample,             // int -- bytes per sample
        (long) record,              // long -- record (1 indexed)
        triggerOffset_samples,      // long -- trigger offset
        samplesThisRead             // U32 -- samples to transfer
        );

    // Process the record segment here
    WriteSamplesToFile(buffer, samplesThisRead);

    // Point to next record segment in on-board memory
    triggerOffset_samples += samplesThisRead;

    // Decrement number of samples left to read
    samplesToRead -= samplesThisRead;
}
```

See "%ATS_SDK_DIR%\Samples\SinglePort\AR_Segments" for a complete sample program that demonstrates how to read records in segments.

2.4.1.3 Using AlazarReadEx

AlazarRead can transfer samples from records acquired to on-board memory that contain up to 2,147,483,647 samples. If a record contains 2,147,483,648 or more samples, use AlazarReadEx rather than AlazarRead.

AlazarReadEx uses signed 64-bit transfer offsets, while AlazarRead uses signed 32-bit transfer offsets. Otherwise, AlazarReadEx and AlazarRead are identical.

2.4.1.4 Using AlazarHyperDisp

HyperDisp technology enables the FPGA on an AlazarTech digitizer board to process sample data. The FPGA divides a record in on-board memory into intervals, finds the minimum and maximum sample values during each interval, and transfers an array of minimum and maximum value pairs to host memory. This allows the acquisition of relatively long records to on-board memory, but the transfer of relatively short processed records across the PCI/PCIe bus to host memory.

For example, an ATS860-256M would require over 2 seconds per channel to transfer 256,000,000 samples across the PCI bus. However, with HyperDisp enabled the ATS860 would require a fraction of a second to calculate HyperDisp data, and transfer a few kilobytes of processed data across the PCI bus. If an application was searching these records for glitches, it may save a considerable amount of time by searching HyperDisp data for the glitches and, if a glitch were found, transfer the raw sample data from the interval from on-board memory to host memory.

Use AlazarHyperDisp to enable a board to process records in on-board memory, and transfer processed records to host memory.

The following code fragment enables an ATS860-256M to process a record in on-board memory containing 250,000,000 samples into an array of 100 HyperDisp points, where each point contains the minimum and maximum sample values over an interval of 2,500,000 samples in the record.

```
// Specify number of samples per record

U32 preTriggerSamples = 125000000;
U32 postTriggerSamples = 125000000;
U32 samplesPerRecord = preTriggerSamples + postTriggerSamples;
U32 recordsPerCapture = 1;

// Acquire to on-board memory (omitted)

// Specify the number of HyperDisp points
U32 pointsPerRecord = 100;

// Allocate a buffer to store the HyperDisp data

U32 bytesPerSample = 1;          // ATS860 constant
U32 samplesPerPoint = 2;         // HyperDisp constant
U32 bytesPerBuffer =
```

```
        bytesPerSample * samplesPerPoint * pointsPerRecord;
U8 *buffer = (U8*) malloc(bytesPerBuffer);

// Enable ATS860 FPGA to process the 250M sample record
// in on-board memory into an array of 100 HyperDisp points,
// and transfer the HyperDisp points into our buffer

U32 error;

AlazarHyperDisp (
    boardHandle,                // HANDLE -- board handle
    NULL,                       // void* -- reserved
    samplesPerRecord,           // U32 -- BufferSize
    (U8*) buffer,               // U8* -- ViewBuffer
    bytesPerBuffer,             // U32 -- ViewBufferSize
    pointsPerRecord,            // U32 -- NumOfPixels
    1,                          // U32 -- Option (1 = HyperDisp)
    CHANNEL_A,                  // U32 -- ChannelSelect
    1,                          // U32 -- record (1 indexed)
    -(long)preTriggerSamples,   // long -- TransferOffset
    &error                      // U32* -- error
    );
```

See "%ATS_SDK_DIR%\Samples\SinglePort\HD" for a complete sample program that demonstrates how to use AlazarHyperDisp.

2.4.1.5 Record timestamps

AlazarTech digitizer boards include a 40-bit counter clocked by the sample clock source scaled by a board specific divider. When a board receives a trigger event to capture a record to on-board memory, it latches and saves the value of this counter. The counter value gives the time, relative to when the counter was reset, when the trigger event for the record occurred.

By default, this counter is reset to zero at the start of each acquisition. Use AlazarResetTimeStamp to control when the record timestamp counter is reset.

Use AlazarGetTriggerAddress to retrieve the timestamp, in timestamp clock ticks, of a record acquired to on-board memory. This function does not convert the timestamp value to seconds.

The following code fragment gets the record timestamp of a record acquired to on-board memory, and converts the timestamp value from clocks ticks to seconds.

```
// Read the record timestamp

U32 triggerAddress;
U32 timestampHigh;
U32 timestampLow;

AlazarGetTriggerAddress (
    boardHandle,            // HANDLE -- board handle
```

```
        record,                 // U32 -- record number (1-indexed)
        &triggerAddress,        // U32* -- trigger address
        &timestampHigh,         // U32* -- timestamp high part
        &timestampLow           // U32* -- timestamp low part
        );

    // Convert the record timestamp from counts to seconds

    __int64 timeStamp_cnt;
    timeStamp_cnt = ((__int64) timestampHigh) << 8;
    timeStamp_cnt |= timestampLow & 0x0ff;

    double samplesPerTimestampCount = 2; // board specific constant
    double samplesPerSec = 50.e6;        // sample rate
    double timeStamp_sec = (double) samplesPerTimestampCount *
        timeStamp_cnt / samplesPerSec;
```

Call AlazarGetParameter with the GET_SAMPLES_PER_TIMESTAMP_CLOCK parameter to obtain the board specific "samples per timestamp count" value. *Table 2-Samples per Timestamp* lists these values.

See "%ATS_SDK_DIR%\Samples\SinglePort\AR_Timestamps" for a complete sample program that demonstrates how to retrieve record timestamps and convert them to seconds.

2.4.1.6 Master-slave applications

If the single-port API is used to acquire from master-slave board system, only the master board in the board system should receive calls to the following API functions: AlazarStartCapture, AlazarAbortCapture, AlazarBusy, AlazarTriggered and AlazarForceTrigger.

See "%ATS_SDK_DIR%\Samples\SinglePort\AR_MasterSlave" for a sample program that demonstrates how to acquire from a master-slave system.

2.4.2 Dual port AutoDMA

AutoDMA allows a board to capture sample data to on-board dual-port memory while – at the same time – transferring sample data from on-board dual-port memory to a buffer in host memory. Data acquisition and data transfer are done in parallel, so any trigger events that occur while the board is transferring data will not be missed.

AutoDMA may be used if:

- A board has dual-port or FIFO on-board memory.
- An application acquires at an average rate, in MB/s, that is less than maximum transfer rate of your board's PCI or PCIe host bus interface.

AutoDMA must be used if:

- A board has FIFO on-board memory.

- An application cannot miss trigger events that occur while it transfers data to host memory, or re-arms for another acquisition.
- An application acquires more sample points or records than can be stored in on-board memory.

Applications such as ultrasonic testing, OCT, radar, and imaging should use AutoDMA.

An AutoDMA acquisition is divided into segments. AutoDMA hardware on a board transfers sample data, one segment at a time, from on-board memory to a buffer in host memory. There may be an unlimited number of segments in an AutoDMA acquisition, so a board can be armed to make an acquisition of infinite duration.

There are four AutoDMA operating modes.

AutoDMA mode	Triggered	Pre-trigger samples	Record headers	Description
Traditional	Yes	Yes	Optional	Acquire multiple records – one per trigger event. Each record may contain samples before and after its trigger event. Each buffer contains one or more records. A record header may optionally precede each record. Supports low trigger repeat rates.
NPT (NoPreTrigger)	Yes	No	No	Acquire multiple records – one per trigger event. Each record may contain only samples after its trigger event. Each buffer contains one or more records. Support high trigger repeat rates.
Triggered streaming	Yes	No	No	Acquire a single, gapless record spanning multiple buffers, where each buffer contains a segment from the record. Wait for a trigger event before acquiring the record.
Continuous streaming	No	No	No	Acquire a single, gapless record spanning multiple buffers, where each buffer contains a segment from the record. Do not wait for a trigger event before acquiring the record.

To make an AutoDMA acquisition, an application must:

- Specify the AutoDMA mode, samples per record, records per buffer, and records per acquisition.

- Arm the board to start the acquisition.
- Wait for an AutoDMA buffer to be filled, process the buffer, and repeat until the acquisition is complete.

The ATS-SDK supplies two groups of functions to make AutoDMA acquisitions: the Asynchronous AutoDMA and Synchronous AutoDMA APIs. Both allow a board to transfer a segment of an AutoDMA acquisition into one buffer while – at the same time – the application processes a previous segment of the acquisition in another buffer.

The following table compares the asynchronous and synchronous AutoDMA APIs.

Attribute	Asynchronous AutoDMA	Synchronous AutoDMA
DMA buffer count	Application defined.	Two API allocated buffers.
CPU usage	Interrupt driven, so very low. More CPU cycles are available to application threads.	Polling loop, so very high. Less CPU cycles are available to application threads.
Data transfer	DMA directly into user-supplied buffer. No CPU cycles are used to copy data.	DMA into API allocated buffer, then copy to user-supplied buffer. CPU cycles used to copy data are not available to application threads.
DMA re-arm time	Next DMA started by hardware interrupt. Latency is lowest and data throughput is highest.	Next DMA started in polling loop. Latency is higher and data throughput is lower.
Master slave systems	Fully supported.	Not recommended.

The synchronous AutoDMA API is deprecated; it is maintained for compatibility with existing applications. The asynchronous AutoDMA API is recommended for all new applications.

2.4.2.1 Traditional AutoDMA

Use traditional mode to acquire multiple records – one per trigger event – with sample points after, and optionally before, the trigger event in each record. A record header may optionally precede each record in the AutoDMA buffer. The programmer specifies the number of samples per record, records per buffer, and buffers in the acquisition.

Traditional AutoDMA supports low trigger repeat rates. For high trigger repeat rates, use NPT AutoDMA mode. Digitizers with four analog input channels do not support 3-channel operation, and require sample interleave to allow for high transfer rates from on-board memory.

Each buffer is organized in memory as follows if a board has on-board memory. Rxy represents a contiguous array of samples from record x of channel y.

Enabled channels	Buffer organization
CH A	R1A, R2A, R3A, … RnA
CH B	R1B, R2B, R3B … RnB
CH A and CH B	R1A, R1B, R2A, R2B, R3A, R3B … RnA, RnB
CH C	R1C, R2C, R3C … RnC
CH A and CH C	R1A, R1C, R2A, R2C, R3A, R3C … RnA, RnC
CH B and CH C	R1B, R1C, R2B, R2C, R3B, R3C … RnB, RnC
CH D	R1D, R2D, R3D … RnD
CH A and CH D	R1A, R1D, R2A, R2D, R3A, R3D … RnA, RnD
CH B and CH D	R1B, R1D, R2B, R2D, R3B, R3D … RnB, RnD
CH C and CH D	R1C, R1D, R2C, R2D, R3C, R3D … RnC, RnD
CH A, CH B, CH C and CH D	R1A, R1B, R1C, R1D, R2A, R2B, R2C, R2D, R3A, R3B, R3C, R3D … RnA, RnB, RnC, RnD

Each buffer is organized in memory as follows if a board does not have on-board memory, or if sample interleave is enabled. Rxy represents a contiguous array of samples from record x of channel y, Rx[uv] represents interleaved samples from record x of channels u and v, and Rx[uvyz] represents interleaved samples from channels u, v, y, and z.

Enabled channels	Buffer organization
CH A	R1A, R2A, R3A, … RnA
CH B	R1B, R2B, R3B … RnB
CH A and CH B	R1[ABAB…], R2[ABAB…], … Rn[ABAB…]
CH C	R1C, R2C, R3C … RnC
CH A and CH C	R1[ACAC…], R2[ACAC…], … Rn[ACAC…]
CH B and CH C	R1[BCBC…], R2[BCBC…], … Rn[BCBC…]
CH D	R1D R2D, R3D … RnD
CH A and CH D	R1[ADAD…], R2[ADAD…], … Rn[ADAD…]
CH B and CH D	R1[BDBD…], R2[BDBD…], … Rn[BDBD…]
CH C and CH D	R1[CDCD…], R2[CDCD…], … Rn[CDCD…]
CH A, CH B, CH C and CH C	R1[ABCDABDC …], R2[ABDCABDC …], … Rn[ABDCABDC…]

See "%ATS_SDK_DIR%\Samples\DualPort\TR" for a sample program that demonstrates how to make an AutoDMA acquisition in Traditional mode.

If record headers are enabled, then a 16-byte record header will precede each record in an AutoDMA buffer. The record header contains a record timestamp, as well as acquisition metadata. See section 2.4.2.5 below for a discussion of AutoDMA record headers.

2.4.2.2 NPT AutoDMA

Use NPT mode to acquire multiple records – one per trigger event – with no sample points before the trigger event in each record, and with no record headers. The programmer specifies the number of samples per record, records per buffer, and buffers in the acquisition.

Note that NPT mode is highly optimized, and supports higher trigger repeats rate than possible in Traditional mode. Digitizers with four analog input channels do not support 3-channel operation, and require sample interleave to allow for high transfer rates from on-board memory.

Each buffer is organized in memory as follows if a board has on-board memory. Rxy represents a contiguous array of samples from record x of channel y.

Enabled channels	Buffer organization
CH A	R1A, R2A, R3A, … RnA
CH B	R1B, R2B, R3B … RnB
CH A and CH B	R1A, R2A, R3A … RnA, R1B, R2B, R3B … RnB
CH C	R1C, R2C, R3C, … RnC
CH A and CH B	R1A, R2A, R3A … RnA, R1B, R2B, R3B … RnB
CH B and CH C	R1B, R2B, R3B … RnB, R1C, R2C, R3C … RnC
CH D	R1D, R2D, R3D, … RnD
CH A and CH D	R1A, R2A, R3A … RnA, R1D, R2D, R3D … RnD
CH B and CH D	R1B, R2B, R3B … RnB, R1D, R2D, R3D … RnD
CH C and CH D	R1C, R2C, R3C … RnC, R1D, R2D, R3D … RnD
CH A, CH B, CH C, and CH D	R1A, R2A, R3A … RnA, R1B, R2B, R3B … RnB, R1C, R2C, R3C … RnC, R1D, R2D, R3D … RnD

Each buffer is organized in memory as follows if a board does not have on-board memory, or if sample interleave is enabled. Rxy represents a contiguous array of samples from record x of channel y, Rx[uv] represents interleaved samples from record x of channels u and v, and Rx[uvyz] represents interleaved samples from record x of channels u, v, y, and z.

Enabled channels	Buffer organization
CH A	R1A, R2A, R3A, … RnA
CH B	R1B, R2B, R3B … RnB
CH A and CH B	R1[ABAB…], R2[ABAB…], … Rn[ABAB…]
CH C	R1C, R2C, R3C … RnC
CH A and CH C	R1[ACAC…], R2[ACAC…], … Rn[ACAC…]
CH B and CH C	R1[BCBC…], R2[BCBC…], … Rn[BCBC…]
CH D	R1D R2D, R3D … RnD
CH A and CH D	R1[ADAD…], R2[ADAD…], … Rn[ADAD…]
CH B and CH D	R1[BDBD…], R2[BDBD…], … Rn[BDBD…]
CH C and CH D	R1[CDCD…], R2[CDCD…], … Rn[CDCD…]

CH A, CH B, CH C and CH D	R1[ABCDABCD …], R2[ABCDABCD …], … Rn[ABCDABCD…]

See "%ATS_SDK_DIR%\Samples\DualPort\NPT" for a sample program that demonstrates how to make an AutoDMA acquisition in NPT mode.

2.4.2.3 Continuous streaming AutoDMA

Use continuous streaming mode to acquire a single, gapless record that spans multiple buffers without waiting for a trigger event to start the acquisition. The programmer specifies the number of samples per buffer, and buffers per acquisition.

Each buffer is organized as follows if a board has on-board memory. R1x represents a contiguous array of samples from channel x.

Enabled channels	Buffer organization
CH A	R1A
CH B	R1B
CH A and CH B	R1A, R1B
CH C	R1C
CH A and CH C	R1A, R1C
CH B and CH C	R1B, R1C
CH D	R1D
CH A and CH D	R1A, R1D
CH B and CH D	R1B, R1D
CH C and CH D	R1C, R1D
CH A, CH B, CH C and CH D	R1A, R1B, R1C, R1D

Each buffer is organized as follows if a board does not have on-board memory, or if sample interleave is enabled. R1x represents a contiguous array of samples from channel x, R1[uv] represents samples interleaved from channels u and v, and R1[uvyz] represents samples interleaved from channels u, v, y, and z.

Enabled channels	Buffer organization
CH A	R1A
CH B	R1B
Both CH A and CH B	R1[ABAB…]
CH C	R1C
CH A and CH C	R1[ACAC…]
CH B and CH C	R1[BCBC…]
CH D	R1D
CH A and CH D	R1[ADAD…]
CH B and CH D	R1[BDBD…]
CH C and CH D	R1[CDCD…]
CH A, CH B, CH C and CH D	R1[ABCDABCD …]

See "%ATS_SDK_DIR%\Samples\DualPort\CS" for a sample program that demonstrates how to make an AutoDMA acquisition in continuous streaming mode.

2.4.2.4 Triggered streaming AutoDMA

Use triggered streaming mode to acquire a single, gapless record that spans two or more buffers after waiting for a trigger event to start the acquisition. The programmer specifies the number of samples in each buffer, and buffers in the acquisition.

Each buffer is organized as follows if a board has on-board memory. R1x represents a contiguous array of samples from channel x.

Enabled channels	Buffer organization
CH A	R1A
CH B	R1B
CH A and CH B	R1A, R1B
CH C	R1C
CH A and CH C	R1A, R1C
CH B and CH C	R1B, R1C
CH D	R1D
CH A and CH D	R1A, R1D
CH B and CH D	R1B, R1D
CH C and CH D	R1C, R1D
CH A, CH B, CH C and CH D	R1A, R2B, R1C, R1D

Each buffer is organized as follows if a board does not have on-board memory, or if sample interleave is enabled. R1x represents a contiguous array of samples from channel x, R1[uv] represents samples interleaved from channels u and v, and R1[uvyz] represents samples interleaved from channels u, v, y, and z.

Enabled channels	Buffer organization
CH A	R1A
CH B	R1B
Both CH A and CH B	R1[ABAB…]
CH C	R1C
CH A and CH C	R1[ACAC…]
CH B and CH C	R1[BCBC…]
CH D	R1D
CH A and CH D	R1[ADAD…]
CH B and CH D	R1[BDBD…]
CH C and CH D	R1[CDCD…]
CH A, CH B, CH C and CH D	R1[ABCDABCD …]

See "%ATS_SDK_DIR%\Samples\DualPort\TS" for a sample program that demonstrates how to make a triggered streaming AutoDMA acquisition.

2.4.2.5 Record headers and timestamps

In traditional AutoDMA mode, a 16-byte record header may optionally precede each record in a buffer.

When record headers are enabled, the following table shows the buffer layout if a board has on-board memory. Record headers are not supported if a board does not have on-board memory. Rxy represents a contiguous array of samples from record x of channel y, and Hxy is a 16-byte record header from record x of channel y.

Enabled channels	Buffer organization
CH A	H1A, R1A, H2A, R2A … HnA, RnA
CH B	H1B, R1B, H2B, R2B … HnB, RnB
CH A and CH B	H1A, R1A, H1B, R1B, H2A, R2A, H2B, R2B… HnA, RnA, HnB, RnB
CH C	H1C, R1C, H2C, R2C … HnC, RnC
CH A and CH C	H1A, R1A, H1C, R1C, H2A, R2A, H2C, R2C… HnA, RnA, HnC, RnC
CH B and CH C	H1B, R1B, H1C, R1C, H2B, R2B, H2C, R2C… HnB, RnB, HnC, RnC
CH D	H1D, R1D, H2D, R2D … HnD, RnD
CH A and CH D	H1A, R1A, H1D, R1D, H2A, R2A, H2D, R2D… HnA, RnA, HnD, RnD
CH B and CH D	H1B, R1B, H1D, R1D, H2B, R2B, H2D, R2D… HnB, RnB, HnD, RnD
CH C and CH D	H1C, R1C, H1D, R1D, H2C, R2C, H2D, R2D… HnC, RnC, HnD, RnD
CH A, CH B, CH C and CH D	H1A, R1A, H1B, R1B, H1C, R1C, H1D, R1D, H2A, R2A, H2B, R2B H2C, R2C, H2D, R2D… HnA, RnA, HnB, RnB, HnC, RnC, HnD, RnD

2.4.2.5.1 Record headers

A record header is a 16-byte structure defined in AlazarApi.h as follows:

```
struct _HEADER0 {
      unsigned int SerialNumber:18;        // bits 17..0
      unsigned int SystemNumber:4;         // bits 21..18
      unsigned int WhichChannel:1;         // bit 22
      unsigned int BoardNumber:4;          // bits 26..23
      unsigned int SampleResolution:3;     // bits 29..27
      unsigned int DataFormat:2;           // bits 31..30
};

struct _HEADER1 {
      unsigned int RecordNumber:24;        // bits 23..0
```

```
      unsigned int BoardType:8;             // bits 31..24
};

struct _HEADER2 {
      U32 TimeStampLowPart;                  //bits 31..0
};

struct _HEADER3 {
      unsigned int TimeStampHighPart:8;     // bits 7..0
      unsigned int ClockSource:2;           // bits 9..8
      unsigned int ClockEdge:1;             // bit 10
      unsigned int SampleRate:7;            // bits 17..11
      unsigned int InputRange:5;            // bits 22..18
      unsigned int InputCoupling:2;         // bits 24..23
      unsigned int InputImpedence:2;        // bits 26..25
      unsigned int ExternalTriggered:1;     // bit 27
      unsigned int ChannelBTriggered:1;     // bit 28
      unsigned int ChannelATriggered:1;     // bit 29
      unsigned int TimeOutOccurred:1;       // bit 30
      unsigned int ThisChannelTriggered:1; // bit 31
};

typedef struct _ALAZAR_HEADER {
      struct _HEADER0 hdr0;
      struct _HEADER1 hdr1;
      struct _HEADER2 hdr2;
      struct _HEADER3 hdr3;
} ALAZAR_HEADER, *PALAZAR_HEADER;
```

A record header contains the following fields:

Field	Width in bits	Description
SerialNumber	18	Serial number of this board as a signed integer.
SystemNumber	4	System identifier number of this board system.
WhichChannel	1	Input channel of this header: 0 = CH A, 1 = CH B.
BoardNumber	4	Board identifier number of this board.
SampleResolution	3	Reserved
DataFormat	2	Reserved
RecordNumber	24	Index of record in acquisition.
BoardType	8	Board type identifier. See AlazarGetBoardKind for a list of board type identifiers.
TimeStampLowPart	32	Lower 32 bits of 40-bit record timestamp. See section 2.4.2.5.2 below.
TimeStampHighPart	8	Upper 8 bits of 40-bit record timestamp. See section 2.4.2.5.2 below.
ClockSource	4	Clock source identifier – 1. See AlazarSetCaptureClock for a list of sample rate identifiers.
ClockEdge	1	Clock edge identifier.

		See AlazarSetCaptureClock for a list of sample rate identifiers.
SampleRate	7	Sample rate identifier. See AlazarSetCaptureClock for a list of sample rate identifiers.
InputRange	5	Input range identifier for this channel. See AlazarInputControl for a list input range identifiers.
InputCoupling	2	Input coupling identifier for this channel. See AlazarInputControl for a list input coupling identifiers.
InputImpedence	2	Input impedance identifier for this channel. See AlazarInputControl for a list input impedance identifiers.
ExternalTriggered	1	This bit is set if TRIG IN on this board caused the board system to trigger and capture this record. Otherwise, this bit is cleared.
ChannelBTriggered	1	This bit is set if CH B on this board caused the board system to trigger and capture this record. Otherwise, this bit is cleared.
ChannelATriggered	1	This bit is set if CH A on this board caused the board system to trigger and capture this record. Otherwise, this bit is cleared.
TimeOutOccurred	1	This bit is set if a trigger timeout expired on a trigger engine on this board caused the board system to trigger and capture this record. Otherwise, this bit is cleared.
ThisChannelTriggered	1	This bit is set if the channel specified by the WhichChannel field on this board caused the board system to trigger and capture this record. Otherwise, this bit is cleared.

See "%ATS_SDK_DIR%\Samples\DualPort\TR_Header" for a full sample program that demonstrates how to make an AutoDMA acquisition in Traditional mode with record headers.

2.4.2.5.2 Record timestamps

AlazarTech digitizer boards include a high-speed 40-bit counter that is clocked by the sample clock source scaled by a board specific divider. When a board receives a trigger event to capture a record to on-board memory, it latches the value of this counter. This timestamp value gives the time, relative to when the counter was reset, when the trigger event for this record occurred.

By default, this counter is reset to zero at the start of each acquisition. Use AlazarResetTimeStamp to control when the record timestamp counter is reset.

The following code fragment demonstrates how to extract the timestamp from a record header, and covert the value from counts to seconds.

```
double samplesPerTimestampCount = 2;    // board specific constant
double samplesPerSec = 100.e6;          // sample rate

void* pRecord;  // points to record header in buffer
ALAZAR_HEADER *pHeader = (ALAZAR_HEADER*) pRecord;

__int64 timestamp_counts;
timestamp_counts = (INT64) pHeader->hdr2.TimeStampLowPart;
timestamp_counts = timestamp_counts |
    (((__int64) (pHeader->hdr3.TimeStampHighPart & 0x0ff)) << 32);

double timestamp_sec = samplesPerTimestampCount *
    timestamp_counts / samplesPerSec;
```

Call AlazarGetParameter with the GET_SAMPLES_PER_TIMESTAMP_CLOCK parameter to determine the board specific "samples per timestamp count" value. *Table 2-Samples per Timestamp* lists these values.

See "%ATS_SDK_DIR%\Samples\DualPort\TR_Header" for a full sample program that demonstrates how to make an AutoDMA acquisition in Traditional mode with record headers, and convert the timestamp to seconds.

2.4.2.6 Using asynchronous AutoDMA

The asynchronous AutoDMA functions allow an application to add user-defined number of buffers to a list of buffers available to be filled by a board, and to wait for the board to receive sufficient trigger events to fill the buffers with sample data. The board uses AutoDMA to transfer data directly into a buffer without making any intermediate copies in memory. As soon as one buffer is filled, the driver automatically starts an AutoDMA transfer into the next available buffer.

2.4.2.6.1 AlazarPostBuffer

C/C++ applications should call AlazarPostAsyncBuffer to make buffers available to be filled by the board, and AlazarWaitAsyncBufferComplete to wait for the board to receive sufficient trigger events to fill the buffers.

The following code fragment outlines the steps required to make an AutoDMA acquisition using AlazarPostAsyncBuffer and AlazarWaitAsyncBufferComplete.

```
// Configure the board to make an AutoDMA acquisition

AlazarBeforeAsyncRead(
    handle,                 // HANDLE -- board handle
    channelMask,            // U32 -- enabled channel mask
    -(long)preTriggerSamples,   // long -- trigger offset
    samplesPerRecord,       // U32 -- samples per record
    recordsPerBuffer,       // U32 -- records per buffer
    recordsPerAcquisition,  // U32 -- records per acquisition
```

```
        flags                   // U32 -- AutoDMA mode and options
        );

    // Add two or more buffers to a list of buffers
    // available to be filled by the board

    for (i = 0; i < BUFFER_COUNT; i++)
    {
        AlazarPostAsyncBuffer(
            handle,             // HANDLE -- board handle
            BufferArray[i],     // void* -- buffer pointer
            BytesPerBuffer      // U32 -- buffer length in bytes
            );
    }

    // Arm the board to begin the acquisition

    AlazarStartCapture(handle);

    // Wait for each buffer in the acquisition to be filled

    U32 buffersCompleted = 0;
    while (buffersCompleted < buffersPerAcquisition)
    {
        // Wait for the board to receives sufficient trigger events
        // to fill the buffer at the head of its list of
        // available buffers.

        U32 bufferIndex = buffersCompleted % BUFFER_COUNT;
        U16* pBuffer = BufferArray[bufferIndex];
        AlazarWaitAsyncBufferComplete(handle, pBuffer, timeout_ms);
        buffersCompleted++;

        // The buffer is full, process it.
        // Note that while the application processes this buffer,
        // the board is filling the next available buffer
        // as trigger events arrive.

        ProcessBuffer(pBuffer, bytesPerBuffer);

        // Add the buffer to the end of the list of buffers
        // available to be filled by this board. The board will
        // fill it with another segment of the acquisition after
        // all of the buffers preceding it have been filled.

        AlazarPostAsyncBuffer(handle, pBuffer, bytesPerBuffer);
    }

    // Abort the acquisition and release resources.
    // This function must be called after an acquisition.

    AlazarAbortAsyncRead(boardHandle);
```

See "%ATS_SDK_DIR%\Samples\DualPort\NPT" for a full sample program that demonstrates make an AutoDMA acquisition using AlazarPostAsyncBuffer.

2.4.2.6.2 ADMA_ALLOC_BUFERS

C#, and LabVIEW applications may find it more convenient to allow the API to allocate and manage a list of buffers available to be filled by the board. These applications should call AlazarBeforeAsyncRead with the AMDA_ALLOC_BUFFERS option selected in the "Flags" parameter.

This option will cause the API to allocate and manage a list of buffers available to be filled by the board. The application must call AlazarWaitNextAsyncBufferComplete to wait for a buffer to be filled. When the board receives sufficient trigger events to fill a buffer, the API will copy the data from the internal buffer to the user-supplied buffer.

The following code fragment outlines how make an AutoDMA acquisition using ADMA_ALLOC_BUFERS flag and AlazarWaitNextAsyncBufferComplete.

```
// Allow the API to allocate and manage AutoDMA buffers

flags |= ADMA_ALLOC_BUFFERS;

// Configure a board to make an AutoDMA acquisition

AlazarBeforeAsyncRead(
    handle,                 // HANDLE -- board handle
    channelMask,            // U32 -- enabled channel mask
    -(long)preTriggerSamples,   // long -- trigger offset
    samplesPerRecord,       // U32 -- samples per record
    recordsPerBuffer,       // U32 -- records per buffer
    recordsPerAcquisition,  // U32 -- records per acquisition
    flags                   // U32 -- AutoDMA mode and options
    );

// Arm the board to begin the acquisition

AlazarStartCapture(handle);

// Wait for each buffer in the acquisition to be filled

RETURN_CODE retCode = ApiSuccess;
while (retCode == ApiSuccess)
{
    // Wait for the board to receive sufficient
    // trigger events to fill an internal AutoDMA buffer.
    // The API will copy data from the internal buffer
    // to the user-supplied buffer.

    retCode =
        AlazarWaitNextAsyncBufferComplete(
            handle,         // HANDLE -- board handle
            pBuffer,        // void* -- buffer to receive data
            bytesToCopy,    // U32 -- bytes to copy into buffer
            timeout_ms      // U32 -- time to wait for buffer
            );

    // The buffer is full, process it
```

```
        // Note that while the application processes this buffer,
        // the board is filling the next available internal buffer
        // as trigger events arrive.

        ProcessBuffer(pBuffer, bytesPerBuffer);
    }

    // Abort the acquisition and release resources.
    // This function must be called after an acquisition.

    AlazarAbortAsyncRead(boardHandle);
```

See "%ATS_SDK_DIR%\Samples\DualPort\CS_WaitNextBuffer" for a full sample program that demonstrates make an AutoDMA acquisition using ADMA_ALLOC_BUFFERS.

An application can get or set the number of DMA buffers allocated by the API by calling AlazarGetParameter or AlazarSetParameter with the parameter SETGET_ASYNC_BUFFCOUNT.

Note that applications may combine ADMA_ALLOC_BUFFERS with options to perform operations that would be difficult in high-level programming languages like LabVIEW. They include:

- Data normalization – This option enables the API to process sample data so that the data always has the same arrangement in the application buffer, independent of AutoDMA mode. See ADMA_GET_PROCESSED_DATA for more information.
- Disk streaming – This option allows the API to use high-performance disk I/O functions to stream buffer data to files. See AlazarCreateStreamFile below for more information.

2.4.2.6.3 AlazarAsyncRead

Some C/C++ applications under Windows may require waiting for an event to be set to the signaled state to indicate when an AutoDMA buffer is full. These applications should use the AlazarAsyncRead API.

The following code fragment outlines how use AlazarAsyncRead to make an asynchronous AutoDMA acquisition.

```
    // Configure the board to make an AutoDMA acquisition

    AlazarBeforeAsyncRead(
        handle,                 // HANDLE -- board handle
        channelMask,            // U32 -- enabled channel mask
        -(long)preTriggerSamples,   // long -- trigger ofset
        samplesPerBuffer,       // U32 -- samples per buffer
        recordsPerBuffer,       // U32 -- records per buffer
        recordsPerAcquisition,  // U32 -- records per acquisition
        admaFlags               // U32 -- AutoDMA flags
        );
```

```
// Add two or more buffers to a list of buffers
// available to be filled by the board

for (i = 0; i < BUFFER_COUNT; i++)
{
    AlazarAsyncRead (
        handle,                          // HANDLE -- board handle
        IoBufferArray[i].buffer,         // void* -- buffer
        IoBufferArray[i].bytesPerBuffer, // U32 -- buffer length
        &IoBufferArray[i].overlapped     // OVERLAPPED*
        );
}

// Arm the board to begin the acquisition

AlazarStartCapture(handle);

// Wait for each buffer in the acquisition to be filled.

U32 buffersCompleted = 0;
while (buffersCompleted < buffersPerAcquisition)
{
    // Wait for the board to receives sufficient
    // trigger events to fill the buffer at the head of its
    // list of available buffers.
    // The event handle will be set to the signaled state when
    // the buffer is full.

    U32 bufferIndex = buffersCompleted % BUFFER_COUNT;
    IO_BUFFER *pIoBuffer = IoBufferArray[bufferIndex];

    WaitForSingleObject(pIoBuffer->hEvent, INFINTE);
    buffersCompleted++;

    // The buffer is full, process it
    // Note that while the application processes this buffer,
    // the board is filling the next available buffer
    // as trigger events arrive.

    ProcessBuffer(pIoBuffer->buffer, pIoBuffer->bytesPerBuffer);

    // Add the buffer to the end of the list of buffers.
    // The board will fill it with another segment from the
    // acquisition after the buffers preceding it have been filled.

    AlazarAsyncRead (
        handle,                      // HANDLE -- board handle
        pIoBuffer->buffer,           // void* -- buffer
        pIoBuffer->bytesPerBuffer,   // U32 -- buffer length
        &pIoBuffer->overlapped       // OVERLAPPED*
        );
}

// Stop the acquisition.
// This function must be called if unfilled buffers are pending.

AlazarAbortAsyncRead(handle);
```

See "%ATS_SDK_DIR%\Samples\DualPort\CS_AsyncRead" for a full sample program that demonstrates make an AutoDMA acquisition using AlazarAsyncRead.

2.4.2.6.4 AlazarAbortAsyncRead

The asynchronous API driver locks application buffers into memory so that boards may DMA directly into them. When a buffer is completed, the driver unlocks it from memory.

An application must call AlazarAbortAsyncRead if, at the end of an acquisition, any of the buffers that it supplies to a board have not been completed. AlazarAbortAsyncRead completes any pending buffers, and unlocks them from memory.

If an application exits without calling AlazarAbortAsyncRead, the API driver may generate a DRIVER_LEFT_LOCKED_PAGES_IN_PROCESS (0x000000CB) bug check error under Windows, or leak the locked memory under Linux.

This may happen, for example, if a programmer runs an application that uses the API under a debugger, stops at a breakpoint, and then stops the debugging session without letting the application or API exit normally.

2.4.2.6.5 Buffer count

An application should supply at least two buffers to a board. This allows the board to fill one buffer while the application consumes the other. As long as the application can consume buffers faster than the board can fill them, the acquisition can continue indefinitely.

However, Microsoft Windows and general-purpose Linux distributions are not real time operating systems. An application thread may be suspended for an indeterminate amount of time to allow other threads with higher priority to run. As a result, buffer processing may take longer than expected.

The board is filling AutoDMA buffers with sample data in real time. If an application is unable to supply buffers as fast a board fills them, the board will run out of buffers into which it can transfer sample data. The board can continue to acquire data until it fills is on-board memory, but then it will abort the acquisition and report a buffer overflow error.

It is recommended that an application supply three or more buffers to a board. This allows some tolerance for operating system latencies. The programmer may need to increase the number of buffers according to the application.

Note that the number of buffers required by a board is not the same as the number of buffers required by an application. There may be little benefit in supplying a board with more than a few tens of buffers, each of a few million samples. If an application requires much more sample data for data analysis or other purposes, the programmer should consider managing application buffers separately from AutoDMA buffers.

2.4.2.7 Using synchronous AutoDMA

Synchronous DMA API assumes that the PCI digitizer being controlled has dual-port acquisition memory.

As shown below, the user program consumes data synchronously with the acquisition loop. Hence the name Synchronous DMA.

A typical sequence of API calls for Synchronous DMA API is shown below. For readability purposes, the following is pseudo-code. Please refer to the sample programs provided for exact syntax and details of what the various parameters passed to these routines mean:

```
// Set up two AutoDMA buffers and start the DMA engine
// Data will be captured in the two buffers in a pin-pong
// mode.  You will be able to process the first buffer while
// data is being captured into the second buffer and
// vice-versa

AlazarStartAutoDMA(h,
                   UserData[0],
                   UseHeader,
                   mode,
                   -(long)bd.PreDepth,
                   transferLength,
                   RecsPerBuffer,
                   bd.RecordCount,
                   &error,
                   CFlags,
                   in1,
                   &r3,
                   &r4);

// Issue Start Capture Command. No data transfer happens before this

AlazarStartCapture( h );


// Wait until all required records have been captured

while (looping == 1)
{
   // Check if one of the AutoDMA buffers has been
   // fully populated or not

   AlazarGetNextAutoDMABuffer(h,
                              UserData[0],
                              UserData[1],
                              &WhichOne,
                              &RecsTransferred ,
                              &error,
                              in1,
                              in1,
                              &TriggersOccurred,
                              &r4);
```

```
    // If WhichOne is equal to 0 or 1, that particular buffer
    // has been populated and hardware is DMAing
    // into the other buffer

    if ((WhichOne == 0)||(WhichOne == 1))
    {
        // Process Your Data here
        // Note that while you process data,
        // new data is still being captured into
        // on-board dual port memory and transferred into
        // the other AutoDMA buffer

        SaveToChannelFiles(UserData[WhichOne]);
    }

    // Check if all records have been captured
    if (RecsTransferred == (long)RecordCount)
    {
        // If all records have been captured, stop the while loop
        looping = 0;
    }
}
```

Note:

- The synchronous AutoDMA API is deprecated, and is maintained for compatibility with existing applications.
- The synchronous AutoDMA API gives poor performance with master-slave systems, and is not recommended for use with such systems.
- Use the CFlags parameter in the call to AlazarStartAutoDMA to select the AutoDMA mode.
- Record headers are only available in Traditional AutoDMA mode. To enable record headers, call AlazarStartAutoDMA with the UseHeader parameter set to 1, and with the mode in the CFlags parameter set to ADMA_TRADITIONAL_MODE.
- AlazarGetNextAutoDMABuffer copies sample data from internally allocated AutoDMA buffers to an application buffer. An application may call this function with a pointer to a single application allocated buffer, rather than two application allocated buffers (Buffer[0] and Buffer[1] above) without affecting AutoDMA operation.
- Calling AlazarWaitNextAsyncBufferComplete in a polling loop is equivalent to calling AlazarEvents, AlazarWaitForBufferReady, and AlazarGetNextAutoDMABuffer, but provides more internally allocated buffers, better throughput, and a simpler programming interface.

2.4.2.8 Scanning applications

Scanning applications divide an acquisition into frames, where each frame is composed of a number of scan lines, and each scan line is composed of a number of sample points.

These applications typically:

- Wait for a "start of frame" event.

- Wait for a number of "start of line" events, capturing a specified number of sample points after each "start of line" event.
- Wait for the next "start of frame" event and repeat.

To implement a scanning application using a hardware "start of frame" signal:

- Connect a TTL signal that will serve as the "start of frame" event to the AUX I/O connector.
- Call AlazarConfigureAuxIO specifying AUX_IN_TRIGGER_ENABLE as the mode, and the active edge of the trigger enable signal as the parameter.
- Configure the board to make an NPT or Traditional mode AutoDMA acquisition where the number of "records per buffer" is equal to the number of scan lines per frame.
- Call AlazarStartCapture to being the acquisition.
- Supply a TTL pulse to the AUX I/O connector (or call AlazarForceTriggerEnable) to arm the board to capture one frame. The board will wait for sufficient trigger events to capture the number of records in an AutoDMA buffer, and then wait for the next trigger enable event.

To implement a scanning application using a software "start of frame" command:

- Call AlazarConfigureAuxIO specifying AUX_OUT_TRIGGER_ENABLE as the mode, along with the signal to output on the AUX I/O connector.
- Configure the board to make an NPT or Traditional mode AutoDMA acquisition where the number of "records per buffer" is equal to the number of scan lines per frame.
- Call AlazarStartCapture to being the acquisition.
- Call AlazarForceTriggerEnable to arm the board to capture one frame. The board will wait for sufficient trigger events to capture the number of records in an AutoDMA buffer, and then wait for the next trigger enable event.

Note that if the number of records per acquisition is set to infinite, software arms the digitizer once to make an AutoDMA acquisition with an infinite number of frames. The hardware will continue acquiring frame data until the acquisition is aborted.

See "%ATS_SDK_DIR%\Samples\DualPort\NPT_Scan" for sample programs that demonstrate how to make a scanning application using hardware trigger enable signals.

If the application configures a digitizer to use an external clock source, and the clock source becomes invalid during horizontal retrace intervals, then the application should consider enabling the "dummy clock" option if supported by the board.

When the "dummy clock" option enabled, the digitizer uses an internally generated clock signal to drive its ADCs for a specified amount of time after the end of each record. At the end of the dummy clock on time, the digitizer switches back to using the external clock signal to clock the ADCs.

The application should configure the dummy clock to be used during the horizontal retrace period after each scan line. See paragraph **Error! Reference source not found.** HYPERLINK \l "_Dummy_clock" Dummy clock for more information.

If an application uses both external clock and external trigger inputs, and the clock source becomes invalid during horizontal retrace intervals, the application should call AlazarSetTriggerOperationForScanning to configure the trigger engines for an external clock source, and to synchronize the start of an acquisition with the next external trigger event after the call to AlazarStartCapture, when the external clock signal is valid. For more information see paragraph **Error! Reference source not found.** HYPERLINK \l "_AlazarSetTriggerOperationForScannin_" AlazarSetTriggerOperationForScanning.

2.4.2.9 Master-slave applications

If a dual-port acquisition API is used to acquire from master-slave board system:

- Call AlazarBeforeAsyncRead on all slave boards before the master board.
- Call AlazarStartCapture only on the master board.
- Call AlazarAbortAsyncRead on the master board before the slave boards.
- The board system acquires the boards in the board system in parallel. As a result, an application must consume a buffer from each board in the board system during each cycle of the acquisition loop.
- Do not use synchronous API functions with master-slave systems – use the asynchronous API functions instead.

The following sample programs demonstrate how to acquire from a master-slave system:
"%ATS_SDK_DIR%\Samples\DualPort\TR_MS",
"%ATS_SDK_DIR%\Samples\DualPort\NPT_MS",
"%ATS_SDK_DIR%\Samples\DualPort\CS_MS", and
"%ATS_SDK_DIR%\Samples\DualPort\TS_MS".

2.4.3 Buffer size and alignment

AlazarTech digitizer boards must be configured to acquire a minimum number of samples per record, and each record must be a multiple of a specified number of samples. Records may shift within a buffer if aligment requirements are not met.

Please refer to *Table 8 - Samples per Record Alignment Requirements* for a list of requirements.

The number of pre-trigger samples in single-port and dual-port "traditional" AutoDMA mode must be a multiple of the pre-trigger aligment value above. See AlazarSetRecordCount and AlazarSetRecordSize for more information.

The address of application buffers passed to the following data transfer functions must meet the buffer aligment requirement in the table above: AlazarRead, AlazarReadEx, AlazarAsyncRead, AlazarPostAsyncBuffer, and AlazarWaitAsyncBufferComplete. For

example, the address of a buffer passed to AlazarPostAsyncBuffer to receive data from an ATS9350 must be aligned to a 32-sample, or 64-byte, address.

Note that AlazarWaitNextAsyncBufferComplete has no aligment requirements. As a result, an application can use this function to transfer data if it is impossible to allocate correctly aligned buffers.

2.4.4 Data format

By default, AlazarTech digitizers generate unsigned sample data. For example, 8-bit digitizers such as the ATS9870 generate sample codes between 0 and 255 (0xFF) where: 0 represents a negative full-scale input voltage, 128 (0x80) represents ~0V input voltage, 255 (0xFF) represents a positive full-scale input voltage.

Some AlazarTech digitizer can be configured to generate signed sample data in two's complement format. For example, the ATS9870 can be configured to generate sample codes where: 0 represents ~0V input voltage, 127 (0x7F) represents a positive full-scale input voltage, and –128 (0x80) represents a negative full-scale input voltage.

Call AlazarSetParameter with parameter SET_DATA_FORMAT before the start of an acquisition to set the sample data format, and call AlazarGetParameter with GET_DATA_FORMAT to get the current data format.

The following code fragment demonstrates how to select signed sample data output.

```
AlazarSetParameter(
    handle,            // HANDLE -- board handle
    0,                 // U8 -- channel Id (not used)
    SET_DATA_FORMAT,   // U32 -- parameter to set
    DATA_FORMAT_SIGNED // long -- value (0 = unsigned, 1 = signed)
    );
```

2.5 Processing data

2.5.1 Converting samples values to volts

The data acquisition API's transfer an array of sample values into an application buffer. Each sample value occupies 1 or 2 bytes in the buffer, where a sample code is stored in the most significant bits of the sample values. Sample values that occupy two bytes are stored with their least significant bytes at the lower byte addresses (little-endian byte order) in the buffer.

To convert sample values in the buffer to volts:

- Get a sample value from the buffer.
- Get the sample code from the most-significant bits of the sample value.
- Convert the sample code to volts.

Note that the arrangement of samples values in the buffer into records and channels depends on the API used to acquire the data.

- Single-port acquisitions return a contiguous array of samples for a specified channel. (See section 2.4.1 "Single-port acquisitions" above.)
- Dual-port AutoDMA acquisitions return sample data whose arrangement depends on the AutoDMA mode and options chosen. (See section 2.4.2 "Dual port AutoDMA" above.)

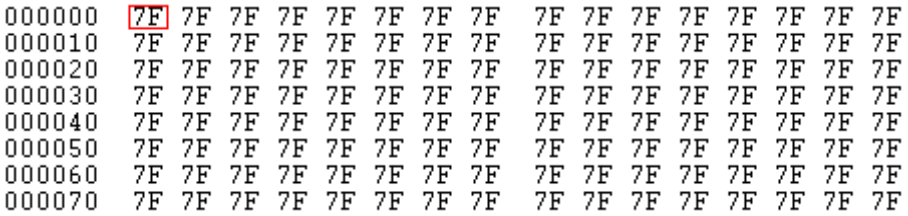
Also note that AlazarTech digitizer boards generate unsigned sample codes by default. (See section 2.4.3 "Data format" above.)

2.5.1.1 8-bits per sample

2.5.1.1.1 Getting 1-byte sample values from the buffer

The figure below shows the first 128-bytes of data in a buffer from an 8-bit digitizer such as the ATS850, ATS860, ATS9850, and ATS9870.

Figure 2-6 8-bit sample data



Each 8-bit sample occupies 1-byte in the buffer, so the figure displays 128 samples (128 bytes / 1 byte per sample).

The following code fragment demonstrates how to access each 8-bit sample value in a buffer.

```
U8 *pSamples = (U8*) buffer;
for (U32 sample = 0; sample < samplesPerBuffer; sample++)
{
   U8 sampleValue = *pSamples++;
   printf("sample value = %02X\n", sampleValue);
}
```

2.5.1.1.2 Getting 8-bit sample codes from 1-byte sample values

Each 8-bit sample value stores an 8-bit sample code. For example, the first byte in buffer above stores the sample code 0x7F, or 127 decimal.

2.5.1.1.3 Converting unsigned 8-bit sample codes to volts

A sample code of 128 (0x80) represents ~0V input voltage, 255 (0xFF) represents a positive full-scale input voltage, and 0 represents a negative full-scale input voltage.

The following table illustrates how unsigned 8-bit sample codes map to values in volts according to the full-scale input range of the input channel.

Sample code in decimal	Sample code in hex	Sample value as a percentage of full-scale input range	Sample value in volts if full-scale input range is ±100 mV	Sample value in volts if full-scale input range is ±1 V
0	0x00	-100%	-100 mV	-1 V
64	0x40	-50%	-50 mV	-.5 V
128	0x80	0%	0 V	0V
192	0xC0	+50%	50 mV	+.5 V
255	0xFF	+100%	+100 mV	+1 V

The following code fragment shows how to convert a 1-byte sample value containing an unsigned 8-bit code to in volts.

```
double SampleToVoltsU8(U8 sampleValue, double inputRange_volts)
{
    // AlazarTech digitizers are calibrated as follows
    int bitsPerSample = 8;
    double codeZero = (1 << (bitsPerSample - 1)) - 0.5;
    double codeRange = (1 << (bitsPerSample - 1)) - 0.5;

    // Convert sample code to volts
    double sampleVolts = inputRange_volts *
       ((double) (sampleValue - codeZero) / codeRange);

    return sampleVolts;
}
```

2.5.1.1.4 Converting signed 8-bit sample codes to volts

A signed code of 0 represents ~0V input voltage, 127 (0x7F) represents a positive full-scale input voltage, and –128 (0x80) represents a negative full-scale input voltage.

The following table illustrates how signed 8-bit sample codes map to values in volts according to the full-scale input range of the input channel.

Sample code in decimal	Sample code in hex	Sample value as a percentage of full-scale input range	Sample value in volts if full-scale input range is ±100 mV	Sample value in volts if full-scale input range is ±1 V
-127	0x81	-100%	-100 mV	-1 V
-64	0xC0	-50%	-50 mV	-.5 V
0	0x00	0%	0 V	0V
64	0x40	+50%	50 mV	+.5 V
127	0x7F	+100%	+100 mV	+1 V

The following code fragment shows how to convert a 1-byte sample value containing a signed 8-bit sample code to in volts.

```
double SampleToVoltsS8(U8 sampleValue, double inputRange_volts)
{
    // AlazarTech digitizers are calibrated as follows
    int bitsPerSample = 8;
    double codeZero = (1 << (bitsPerSample - 1)) - 0.5;
    double codeRange = (1 << (bitsPerSample - 1)) - 0.5;

    // Convert signed code to unsigned
    U8 sampleCode = sampleValue + 0x80;

    // Convert sample code to volts
    double sampleVolts = inputRange_volts *
       ((double) (sampleCode - codeZero) / codeRange);

    return sampleVolts;
}
```

2.5.1.2 12-bits per sample

2.5.1.2.1 Getting 2-byte sample values from the buffer

The figure below displays the first 128-bytes of data in a buffer from a 12-bit digitizer such as the ATS310, ATS330, ATS9325, ATS9350, ATS9351, ATS9360, and ATS9373.

Figure 2-7 12-bit sample data

```
000000  E0 7F F0 7F 00 80 F0 7F  F0 7F 10 80 E0 7F 00 80
000010  F0 7F 00 80 E0 7F E0 7F  00 80 E0 7F F0 7F F0 7F
000020  00 80 F0 7F 00 80 D0 7F  F0 7F E0 7F E0 7F 00 80
000030  F0 7F F0 7F 00 80 E0 7F  00 80 F0 7F F0 7F 00 80
000040  F0 7F F0 7F E0 7F E0 7F  10 80 00 80 E0 7F 10 80
000050  00 80 E0 7F E0 7F F0 7F  E0 7F 10 80 F0 7F 00 80
000060  00 80 D0 7F F0 7F F0 7F  E0 7F 00 80 00 80 F0 7F
000070  00 80 F0 7F 00 80 E0 7F  00 80 F0 7F F0 7F E0 7F
```

Each 12-bit sample value occupies a 2-bytes in the buffer, so the figure displays 64 sample values (128 bytes / 2 bytes per sample).

The first 2 bytes in the buffer, shown highlighted, are 0xE0 and 0x7F. Two-byte sample values are stored in little-endian byte order in the buffer, so the first sample value in the buffer is 0x7FE0.

The following code fragment demonstrates how to access each 16-bit sample value in a buffer.

```
U16 *pSamples = (U16*)buffer;
for (U32 sample = 0; sample < samplesPerBuffer; sample++)
{
   U16 sampleValue = *pSamples++;
   printf("sample value = %04X\n", sampleValue);
}
```

© 2003-2015 Alazar Technologies Inc. Version 7.1.1

2.5.1.2.2 Getting 12-bit sample codes from 16-bit sample values

A12-bit sample code is stored in the most significant bits of each 16-bit sample value, so right-shift each 16-bit value by 4 (or divide by 16) to obtain the 12-bit sample code. In the example above, the 16-bit sample value 0x7FE0 right-shifted by four results in the 12-bit sample code 0x7FE, or 2046 decimal.

16-bit sample value in decimal	32736
16-bit sample value in hex	7FE0
16-bit sample value in binary	0111 1111 1110 0000
12-bit sample code from most-significant bits of 16-bit sample value	0111 1101 1110
12-bit sample code in hex	7FE
12-bit sample code in decimal	2046

2.5.1.2.3 Converting unsigned 12-bit sample codes to volts

An unsigned code of 2048 (0x800) represents ~0V input voltage, 4095 (0xFFF) represents a positive full-scale input voltage, and 0 represents a negative full-scale input voltage.

The following table illustrates how unsigned 12-bit sample codes map to values in volts according to the full-scale input range of the input channel.

Sample code in decimal	Sample code in hex	Sample value as a percentage of full-scale input range	Sample value in volts if full-scale input range is ±100 mV	Sample value in volts if full-scale input range is ±1 V
0	0x000	-100%	-100 mV	-1 V
1024	0x400	-50%	-50 mV	-.5 V
2048	0x800	0%	0 V	0V
3072	0xC00	+50%	50 mV	+.5 V
4095	0xFFF	+100%	+100 mV	+1 V

The following code fragment demonstrates how to convert a 2-byte word containing an unsigned 12-bit sample code to in volts.

```
double SampleToVoltsU12(U16 sampleValue, double inputRange_volts)
{
    // Right-shift 16-bit sample word by 4 to get 12-bit sample code
    int bitShift = 4;
    U16 sampleCode = sampeValue >> bitShift;

    // AlazarTech digitizers are calibrated as follows
    int bitsPerSample = 12;
    double codeZero = (1 << (bitsPerSample - 1)) - 0.5;
    double codeRange = (1 << (bitsPerSample - 1)) - 0.5;

    // Convert sample code to volts
```

```
    double sampleVolts = inputRange_volts *
       ((double) (sampleCode - codeZero) / codeRange);

    return sampleVolts;
}
```

2.5.1.2.4 Converting signed 12-bit sample codes to volts

A signed code of 0 represents ~0V input voltage, 2047 (0x7FF) represents a positive full-scale input voltage, and -2048 (0x800) represents a negative full-scale input voltage.

The following table illustrates how signed 12-bit sample codes map to values in volts according to the full-scale input range of the input channel.

Sample code in decimal	Sample code in hex	Sample value as a percentage of full-scale input range	Sample value in volts if full-scale input range is ±100 mV	Sample value in volts if full-scale input range is ±1 V
-2047	0x801	-100%	-100 mV	-1 V
-1024	0xC00	-50%	-50 mV	-.5 V
0	0x000	0%	0 V	0V
1024	0x400	+50%	50 mV	+.5 V
2047	0x7FF	+100%	+100 mV	+1 V

The following code fragment shows how to convert a 2-byte sample word containing a signed 12-bit sample code to in volts.

```
double SampleToVoltsS12(U16 sampleValue, double inputRange_volts)
{
    // Right-shift 16-bit sample value by 4 to get 12-bit sample code
    int bitShift = 4;
    U16 sampleCode = sampleValue >> bitShift;

    // Convert signed code to unsigned
    sampleCode = (sampleCode + 0x800) & 0x7FF;

    // AlazarTech digitizers are calibrated as follows
    int bitsPerSample = 12;
    double codeZero = (1 << (bitsPerSample - 1)) - 0.5;
    double codeRange = (1 << (bitsPerSample - 1)) - 0.5;

    // Convert sample code to volts
    double sampleVolts = inputRange_volts *
       ((double) (sampleCode - codeZero) / codeRange);

    return sampleVolts;
}
```

2.5.1.3 14-bits per sample

2.5.1.3.1 Getting 2-byte sample values from the buffer

The figure below displays the first 128-bytes of data in a buffer from a 14-bit digitizer such as the ATS460 and ATS9440.

Figure 2-8 14-bit sample data

```
000000  4C 7F EC 7F 3C 80 98 80  D0 80 24 81 7C 81 B4 81
000010  3C 82 B4 82 A8 82 60 83  9C 83 14 84 40 84 88 84
000020  E0 84 50 85 D0 85 FC 85  2C 86 B0 86 10 87 58 87
000030  AC 87 10 88 38 88 CC 88  30 89 94 89 98 89 30 8A
000040  44 8A D8 8A 64 8B C8 8B  B4 8B 34 8C 78 8C B0 8C
000050  28 8D 94 8D D8 8D 40 8E  90 8E C4 8E 6C 8F 3C 8F
000060  04 90 C8 8F 40 8F E8 8E  B0 8E 74 8E E0 8D B4 8D
000070  48 8D E8 8C 98 8C 2C 8C  F4 8B 70 8B 3C 8B FC 8A
```

Each sample value occupies a 2-bytes in the buffer, so the figure displays 64 sample values (128 bytes / 2 bytes per sample).

The first 2 bytes in the buffer, shown highlighted, are 0x4C and 0x7F. Two-byte sample values are stored in little-endian byte order in the buffer, so the first sample value in the buffer is 0x7F4C.

The following code fragment demonstrates how to access each 16-bit sample value in a buffer.

```
U16 *pSamples = (U16*) buffer;
for (U32 sample = 0; sample < samplesPerBuffer; sample++)
{
   U16 sampleValue = *pSamples++;
   printf("sample value = %04X\n", sampleValue);
}
```

2.5.1.3.2 Getting 14-bit sample codes from 16-bit sample values

A 14-bit sample code is stored in the most significant bits of each 16-bit sample value in the buffer, so right-shift each 16-bit value by 2 (or divide by 4) to obtain the 14-bit sample code. In the example above, the 16-bit value 0x7F4C right-shifted by two results in the 14-bit sample code 0x1FD3, or 8147 decimal.

16-bit sample value in decimal	32588
16-bit sample value in hex	7F4C
16-bit sample value in binary	0111 1111 0100 1100
14-bit sample code from most-significant bits of 16-bit sample value	01 1111 1101 0011
14-bit sample code in hex	1FD3
14-bit sample code in decimal	8147

2.5.1.3.3 Converting unsigned 14-bit sample codes to volts

An unsigned code of 8192 (0x2000) represents ~0V input voltage, 16383 (0x3FFF) represents a positive full-scale input voltage, and 0 represents a negative full-scale input voltage.

The following table illustrates how unsigned 14-bit sample codes map to values in volts according to the full-scale input range of an input channel.

Sample code in decimal	Sample code in hex	Sample value as a percentage of full-scale input range	Sample value in volts if full-scale input range is ±100 mV	Sample value in volts if full-scale input range is ±1 V
0	0x0000	-100%	-100 mV	-1 V
4096	0x1000	-50%	-50 mV	-.5 V
8192	0x2000	0%	0 V	0V
12288	0x3000	+50%	50 mV	+.5 V
16383	0x3FFF	+100%	+100 mV	+1 V

The following code fragment demonstrates how to convert a 2-byte sample value containing an unsigned 14-bit sample code to in volts.

```
double SampleToVoltsU14(U16 sampleValue, double inputRange_volts)
{
    // Right-shift 16-bit sample word by 2 to get 14-bit sample code
    int bitShift = 2;
    U16 sampleCode = sampleValue >> bitShift;

    // AlazarTech digitizers are calibrated as follows
    int bitsPerSample = 14;
    double codeZero = (1 << (bitsPerSample - 1)) - 0.5;
    double codeRange = (1 << (bitsPerSample - 1)) - 0.5;

    // Convert sample code to volts
    double sampleVolts = inputRange_volts *
       ((double) (sampleCode - codeZero) / codeRange);

    return sampleVolts;
}
```

2.5.1.3.4 Converting signed 14-bit sample codes to volts

A signed code of 0 represents ~0V input voltage, 8191 (0x1FFF) represents a positive full-scale input voltage, and –8192 (0x2000) represents a negative full-scale input voltage.

The following table illustrates how signed 14-bit sample codes map to values in volts depending on the full-scale input range of the input channel.

Sample code in decimal	Sample code in hex	Sample value as a percentage of full-scale input range	Sample value in volts if full-scale input range is	Sample value in volts if full-scale input

			±100 mV	range is ±1 V
-8191	0x2001	-100%	-100 mV	-1 V
-4096	0x3000	-50%	-50 mV	-.5 V
0	0x0000	0%	0 V	0V
4096	0x1000	+50%	50 mV	+.5 V
8191	0x1FFF	+100%	+100 mV	+1 V

The following code fragment demonstrates how to convert a 2-byte sample value containing a signed 14-bit sample code to in volts.

```
double SampleToVoltsU14(U16 sampleValue, double inputRange_volts)
{
    // Right-shift 16-bit sample word by 2 to get 14-bit sample code
    int bitShift = 2;
    U16 sampleCode = sampeWord >> bitShift;

    // AlazarTech digitizers are calibrated as follows
    int bitsPerSample = 14;
    double codeZero = (1 << (bitsPerSample - 1)) - 0.5;
    double codeRange = (1 << (bitsPerSample - 1)) - 0.5;

    // Convert the signed code to unsigned
    sampleCode = (sampleCode + 0x2000) & 0x1FFF;

    // Convert sample code to volts
    double sampleVolts = inputRange_volts *
       ((double) (sampleCode - codeZero) / codeRange);

    return sampleVolts;
}
```

2.5.1.4 16-bit per sample

2.5.1.4.1 Getting 2-byte sample values from the buffer

The figure below displays the first 128-bytes of data in a buffer from a 16-bit digitizer such as the ATS660, ATS9462, ATS9625, or ATS9626.

Figure 2-9 16-bit sample data

```
000000  14 80 FB 7F FB 7F 08 80  FB 7F 00 80 02 80 ED 7F
000010  0B 80 FF 7F F8 7F 0B 80  09 80 0E 80 F3 7F FE 7F
000020  FF 7F E7 7F FA 7F 15 80  F8 7F EC 7F 0C 80 EF 7F
000030  EF 7F F1 7F F5 7F 06 80  FA 7F 02 80 E2 7F FC 7F
000040  17 80 FB 7F 04 80 0A 80  08 80 00 80 07 80 0B 80
000050  FF 7F 04 80 06 80 04 80  04 80 0B 80 F8 7F FC 7F
000060  F5 7F F7 7F 18 80 09 80  03 80 00 80 F8 7F 0A 80
000070  F2 7F F4 7F F3 7F F0 7F  FC 7F FA 7F 01 80 EF 7F
```

Each 16-bit sample value occupies 2 bytes in the buffer, so the figure displays 64 sample values (128 bytes / 2 bytes per sample).

The first 2 bytes in the buffer, shown highlighted, are 0x14 and 0x80. Two-byte samples values are stored in little-endian byte order in the buffer, so the first sample value is 0x8014.

The following code fragment demonstrates how to access each 16-bit sample value in a buffer.

```
U16 *pSamples = (U16*)buffer;
for (U32 sample = 0; sample < samplesPerBuffer; sample++)
{
   U16 sampleValue = * pSamples++;
   printf("sample value = %04X\n", sampleValue);
}
```

2.5.1.4.2 Getting 16-bit sample codes from 16-bit sample values

A16-bit sample code is stored in each 16-bit sample value in the buffer. In the example above, the first sample code is 0x8014, or 32788 decimal.

2.5.1.4.3 Converting unsigned 16-bit sample codes to volts

An unsigned code of 32768 (0x8000) represents ~0V input voltage, 65535 (0xFFFF) represents a positive full-scale input voltage, and 0 represents a negative full-scale input voltage.

The following table illustrates how unsigned 16-bit sample codes map to values in volts according to the full-scale input range of an input channel.

Sample code in decimal	Sample code in hex	Sample value as a percentage of full-scale input range	Sample value in volts if full-scale input range is ±100 mV	Sample value in volts if full-scale input range is ±1 V
0	0x0000	-100%	-100 mV	-1 V
16384	0x4000	-50%	-50 mV	-.5 V
32768	0x8000	0%	0 V	0V
49152	0xC000	+50%	50 mV	+.5 V
65535	0xFFFF	+100%	+100 mV	+1 V

The following code fragment demonstrates how to convert a 2-byte sample value containing an unsigned 16-bit sample code to in volts.

```
double SampleToVoltsU16(U16 sampleValue, double inputRange_volts)
{
    // AlazarTech digitizers are calibrated as follows
    int bitsPerSample = 16;
    double codeZero = (1 << (bitsPerSample - 1)) - 0.5;
    double codeRange = (1 << (bitsPerSample - 1)) - 0.5;

    // Convert sample code to volts
    double sampleVolts = inputRange_volts *
       ((double) (sampleValue - codeZero) / codeRange);
```

```
    return sampleVolts;
}
```

2.5.1.4.4 Converting signed 16-bit sample codes to volts

A signed code of 32767 (0x7FFF) represents a positive full-scale input voltage, 0 represents ~0V input voltage, and –32768 (0x8000) represents a negative full-scale input voltage.

The following table illustrates how signed 16-bit sample codes map to values in volts according to the full-scale input range of the input channel.

Sample code in decimal	Sample code in hex	Sample value as a percentage of full-scale input range	Sample value in volts if full-scale input range is ±100 mV	Sample value in volts if full-scale input range is ±1 V
-32767	0x8001	-100%	-100 mV	-1 V
-16384	0xC000	-50%	-50 mV	-.5 V
0	0x0000	0%	0 V	0V
16384	0x4000	+50%	50 mV	+.5 V
32767	0x7FFF	+100%	+100 mV	+1 V

The following code fragment demonstrates how to convert a 2-byte sample word containing a signed 16-bit sample code to in volts.

```
double SampleToVoltsS16(U16 sampleValue, double inputRange_volts)
{
    // AlazarTech digitizers are calibrated as follows
    int bitsPerSample = 16;
    double codeZero = (1 << (bitsPerSample - 1)) - 0.5;
    double codeRange = (1 << (bitsPerSample - 1)) - 0.5;

    // Convert signed sample value to unsigned code
    U16 sampleCode = (sampleValue + 0x8000);

    // Convert sample code to volts
    double sampleVolts = inputRange_volts *
       ((double) (sampleCode - codeZero) / codeRange);

    return sampleVolts;
}
```

2.5.2 Saving binary files

If an application saves sample data to a binary data file for later processing, it may be possible to improve disk write speeds by considering the following recommendations.

2.5.2.1 C/C++ applications

If the application is written in C/C++ and is running under Windows, use the Windows CreateFile API with the FILE_FLAG_NO_BUFFERING flag for file I/O, if possible. Sequential disk write speeds are often substantially higher when this option is selected.

See "%ATS_SDK_DIR%\Samples\DualPort\TS_DisableFileCache" for a sample program that demonstrates how to use this API to stream data to disk.

2.5.2.2 LabVIEW applications

If the application is written in LabVIEW, or another high-level programming environment, then consider using the AlazarCreateStreamFile API function. This function creates a binary data file, and enables the API to save each buffer received during an AutoDMA acquisition to this file.

The API uses high-performance disk I/O functions that would be difficult to implement in high-level environments like LabVIEW. As a result, it allows an application in such an environment to perform high-performance disk streaming with a single additional function call.

The following code fragment outlines how to write a disk streaming application using AlazarCreateStreamFile:

```
// Allow the API to allocate and manage AutoDMA buffers

flags |= ADMA_ALLOC_BUFFERS;

// Configure the board to make an AutoDMA acquisition

AlazarBeforeAsyncRead(
    handle,                 // HANDLE -- board handle
    channelMask,            // U32 -- enabled channel mask
    -(long)preTriggerSamples,   // long -- trigger offset
    samplesPerRecord,       // U32 -- samples per record
    recordsPerBuffer,       // U32 -- records per buffer
    recordsPerAcquisition,  // U32 -- records per acquisition
    flags                   // U32 -- AutoDMA mode and options
    );

// Create a binary data file, and enable the API save each
// AutoDMA buffer to this file.

AlazarCreateStreamFile(handle, "data.bin");

// Arm the board to begin the acquisition

AlazarStartCapture(handle);

// Wait for each buffer in the acquisition to be filled

RETURN_CODE retCode = ApiSuccess;
while (retCode == ApiSuccess)
```

```
{
    // Wait for the board to receive sufficient trigger
    // events to fill an internal buffer.
    // The API will save the buffer to a binary data file,
    // but will not copy any data into our buffer.

    retCode =
        AlazarWaitNextAsyncBufferComplete(
            handle,         // HANDLE -- board handle
            NULL,           // void* -- buffer to receive data
            0,              // U32 -- bytes to copy into buffer
            timeout_ms      // U32 -- time to wait for buffer
            );
}

// Abort the acquisition and release resources.
// This function must be called after an acquisition.

AlazarAbortAsyncRead(boardHandle);
```

See "%ATS_SDK_DIR%\Samples\DualPort\CS_CreateStreamFile" for a full sample program that demonstrates how to stream sample data to disk using AlazarCreateStreamFile.

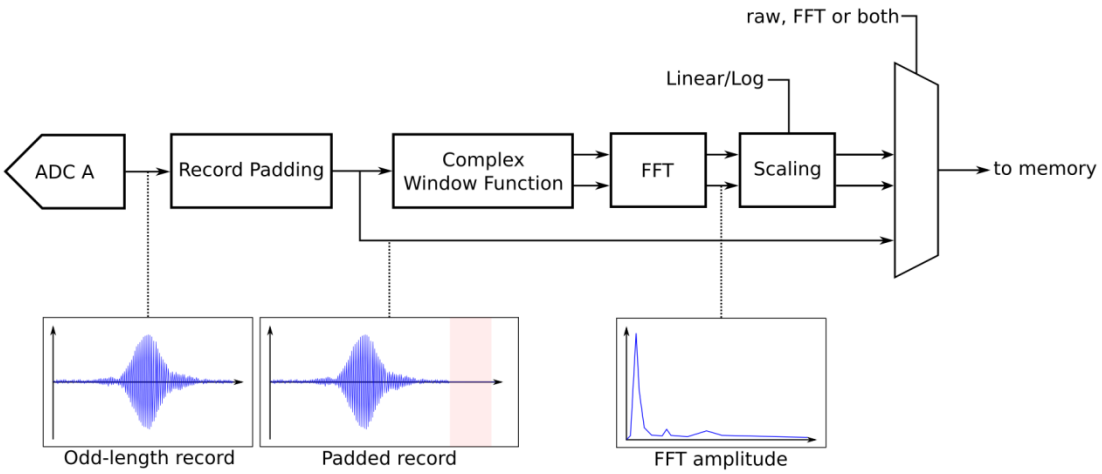
2.6 AlazarDSP

This section presents the AlazarDSP API that allows accessing the on-FPGA DSP features provided with some AlazarTech digitizers. Knowledge of the ATS-SDK API as documented in the previous sections of this document is required to take full advantage of the information presented here.

2.6.1 On-FPGA FFT Overview

The first DSP module to make it into AlazarDSP is a Fast Fourier Transform block implemented in ATS9350, ATS9360 and ATS9373.

When this module is active, data must be captured from input channel A only. The acquired records can be padded then mutiplied with a complex window function before going in the FFT processing block. The resulting data can optionaly be scaled to get its log. The nature of the output data can be chosen (amplitude, real, imaginary), and it is then possible to set the output format from a variety of combinations (floating point, 32-bit unsigned integer, etc.). Lastly, it is possible to get at the output either FFT data, raw data or both.



2.6.2 General Programming Concepts

All the functions from the AlazarDSP module are defined in AlazarDSP.h, and are implemented in the usual ATSApi library (ATSApi.dll under Windows, and libATSApi.so under Linux).

Function are prefixed either with AlazarDSP if they apply to any DSP block, or by AlazarFFT if they are specific to fast Fourier transform modules.

The AlazarDSP API introduces a new type called dsp_module_handle, which represents a DSP module within a digitizer. Depending on their scope, function calls either require a board or a DSP module handle to be passed.

The AlazarDSP functions must be used in the context of AutoDMA NPT applications.

2.6.3 Program Flow

This section details all the steps that are required to take a working AutoDMA NPT program and turn it into a FFT program.

Function calls to the AlazarTech API are usually split into two categories: board configuration and data acquisition. This is best seen in the code samples provided with the ATS-SDK, where this separation is shown by sub-routines. Most of the AlazarDSP function calls fall into the second category. This means that by and large, the board configuration routine of the existing code samples is left untouched.

Programs that use the AlazarDSP API need to get the handle of the DSP module they want to use. This is done by calling AlazarDSPGetModules(). Information about the DSP module can be retrieved at any time using AlazarDSPGetInfo().

The board configuration section is left untouched when compared to a standard AutoDMA NPT acquisition.
In the data acquisition section, the following changes must be made:

1. AlazarSetRecordSize() is not called. This function is called internally by AlazarSetupFFT().
2. AlazarSetupFFT() is called before AlazarBeforeAsyncRead() and before allocating the DMA buffers. This is due to the fact that AlazarSetupFFT() returns the effective number of bytes per record that need to be allocated and passed to AlazarBeforeAsyncRead()
3. AlazarBeforeAsyncRead() is called by passing:
    a. The number of **bytes per record** to the fourth parameter (SamplesPerRecord)
    b. 0x7FFFFFFF to RecordsPerAcquisition
    c. OR'ing ADMA_DSP to the existing auto-dma flags
4. AlazarWaitAsyncBufferComplete() is replaced with AlazarDSPGetBuffer().
5. AlazarAbortAsyncRead() is replaced with AlazarDSPAbortCapture().

3 Advanced Topics

3.1 External Clock Issues for OCT applications

The external clocking feature of AlazarTech boards is commonly used in OCT applications, where swept laser sources generate a signal to be used for clocking the acquisition. However, in some cases the external clock signal does not meet the requirements of the digitizers, which can lead to various issues. This section discusses the steps that need to be taken to diagnose and troubleshoot external clock problems.

3.1.1 Diagnose External Clock Issues

External clock issues can be of two natures; trigger jumps, or unexpected (glitchy) acquired data. These issues can also arise as the result of a board misconfiguration (bad record length, bad trigger configuration…). Before proceeding with the external clock troubleshooting, you *must* ensure that the external clock is indeed the cause of your problems. One way to do that is to make sure that your acquisition works fine when using the internal clock. Another way is to reproduce your acquisition configuration in AlazarDSO, and make sure that the problem also shows up there.

Once having made sure that the external clock is the issue, the next step is to identify the problematic regions of the signal. To do this, please acquire a few record acquisition cycles (laser sweeps) with a high speed oscilloscope (ideally 20GS/s, 4GHz), and to send the results to us.

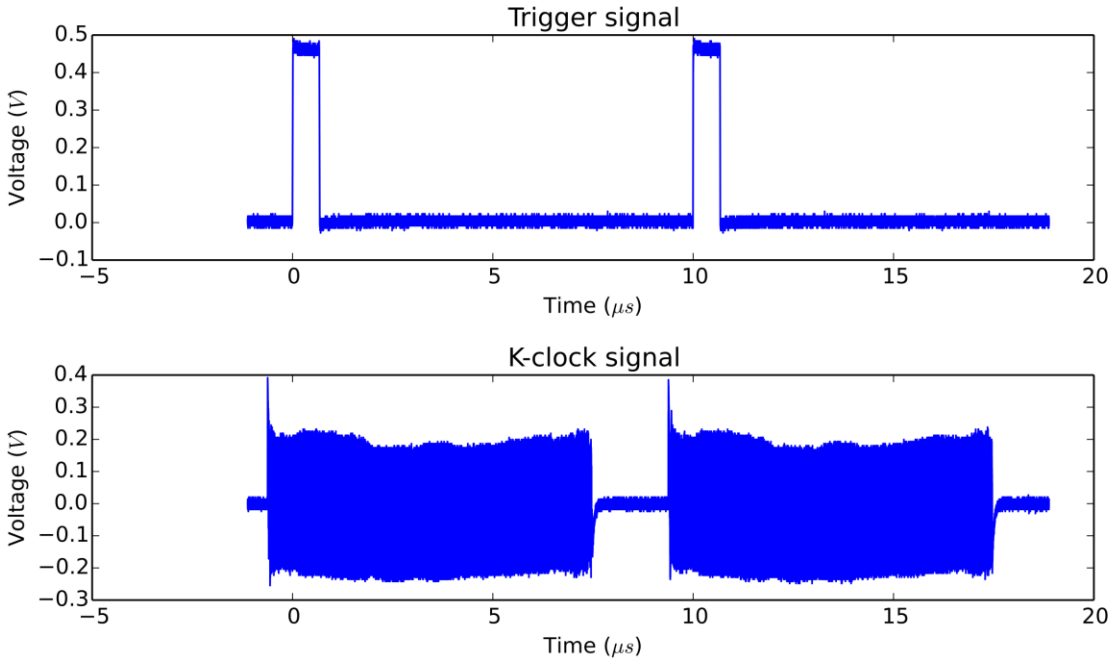


Figure 10 - External Clock Measurement

Here is an example of an external clock analysis plot, annotated to show the problem:

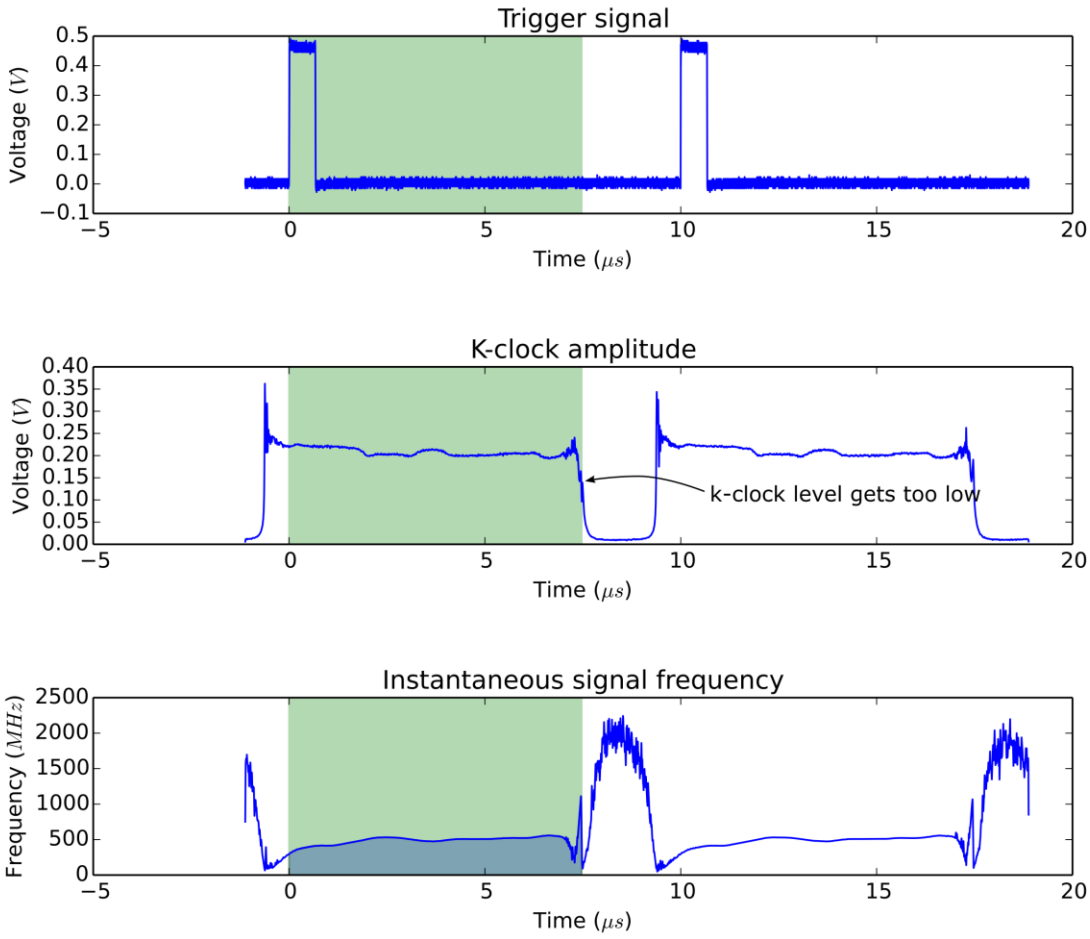


Figure 11- Example of external clock analysis

3.1.2 K-clock deglitching firmware

The k-clock deglitching firmware available for ATS9350 and ATS9351 is specifically designed to overcome k-clock related issues. If you are using one of these boards, trying this firmware is the next logical step. In our experience, it solves all k-clock related issues. ATS9360 and ATS9373's firmwares include the deglitching feature by default. Dummy clockdummy clock should be tried for the ATS9350 and ATS9351 boards only in cases where the deglitching firmware does not solve the issue.

When the "dummy clock" option is enabled, the digitizer uses an internally generated clock signal to drive its ADCs for a specified amount of time after the end of each record. At the end of the dummy clock on time, the digitizer switches back to using the external clock signal to clock the ADCs.

Call AlazarSetClockSwitchOver to enable dummy clock mode, and set the amount of tiem to use the dummy clock after the end of each record.

```
AlzarSetClockSwithOver(
   handle,                    // HANDLE -- board handle
   CSO_ENABLE_DUMMY_CLOCK,    // U32 -- mode
   dummyClockOnTime_ns,       // U32 -- dummy clock on time in ns
   0                          // U32 -- reserved
   );
```

Note that the on-time should be selected so that it is longer than the amount time that external clock signal is unstable, but not longer than the time between the end of one scanline and start of the next scan-line.

3.2 AlazarSetTriggerOperationForScanning

AlazarTech digitizers require that the ADC clock be valid when an application calls AlazarStartCapture to arm a board to begin an acquisition. The digitizer may not be able to start an acquisition if the the application calls AlazarStartCapture while the ADC clock is invalid

If an application uses both external clock and external trigger signals, and the external clock is not suitable to drive the ADC's during part of the interval between trigger events, the application can call AlazarSetTriggerOperationForScanning (rather than AlazarSetTriggerOperation) to configure the trigger engines. This function configures the trigger engines to use an external trigger source connected to the TRIG IN connector, and also allows the board to begin an acquisition on the next external trigger event after the call to AlazarStartCapture, when the external clock signal is valid.

For example, some OCT applications use a laser source that supplies an external clock signal that is valid on the rising edge of the trigger pulse, but falls to 0 Hz on the falling edge of the trigger pulse. The digitizer may not work correctly if the application calls AlazarStartCapture to arm the board while the clock output is at 0 Hz. These applications can call AlazarSetTriggerOperationForScanning to configure the trigger engines to use an external trigger input, and to wait until the first rising edge of the external trigger pulse arrives after the call the AlazarStartCapture to start the acquisition, when the external clock is valid.

```
RETURN_CODE
AlazarSetTriggerOperationForScanning (
   HANDLE handle,
   U32 SlopeId,            // trigger slope identifier
   U32 Level,              // trigger level code
   U32 Options             // scanning options
   );
```

AlazarSetTriggerOperationForScanning configures a board to use trigger operation TRIG_ENGINE_OP_J, and configures the source of TRIG_ENGINE_J to be TRIG_EXTERNAL. The application must call AlazarSetExternalTrigger to set the full-

scale external input range and coupling of the external trigger signal connected to the TRIG IN connector.

The slope identifier parameter selects if a trigger event should be generated when the external trigger level rise above, or falls below, a specified level. The parameter may have one of the following values.

Identifier	Value	Description
TRIGGER_SLOPE_POSITIVE	1	The external trigger level rises above a specified level.
TRIGGER_SLOPE_NEGATIVE	2	The external trigger level falls below a specified level.

The trigger level parameter sets the external trigger level as an unsigned 8-bit code that represents a fraction of the external trigger full scale input range: 0 represents the negative full-scale input, 128 represents a 0 volt input, and 255 represents the positive full-scale input.

In general, the trigger level value is given by:

```
TriggerLevelCode = 128 + 127 * TriggerLevelVolts / InputRangeVolts
```

The following table gives examples of how trigger level codes map to trigger levels in volts according to the external trigger full-scale input range.

Trigger level code	Trigger level as fraction of external input range	Trigger level in ±1V external trigger range	Trigger level in ±5V external trigger input range
0	-100%	-1V	-5V
64	-50%	-500 mV	-2.5 V
96	-25%	-250 mV	-1.25 V
128	0%	0 V	0 V
160	+25 %	250 mV	1.25 V
192	+50%	+500 mV	+2.5 V
255	+100%	+1V	+5V

The options parameter may be one of the following flags:

Identifier	Meaning
STOS_OPTION_DEFER_START_CAPTURE (0x00000001)	Wait until the next external trigger event after the application calls AlazarStartCapture before arming the board to start the acquisition. The external clock input should be valid when the trigger event arrives.

4 Reference

4.1 Function Groups

The ATS-SDK is organized into the following functional groups. See AlazarApi.h for function declarations.

4.1.1 Initialization

Name	Purpose
AlazarBoardsFound	Get the number of boards detected in all board systems.
AlazarBoardsInSystemByHandle	Get the number of boards in the board system specified by the handle to its master board.
AlazarBoardsInSystemBySystemID	Get the number of boards in the board system specified by its system identifier.
AlazarClose	Close a board handle.
AlazarGetBoardBySystemHandle	Get a handle to a board specified by its board identifier and handle to the master board in its board system.
AlazarGetBoardBySystemID	Get a handle to a board specified by its system identifier and board identifier.
AlazarGetSystemHandle	Get a handle to the master board in a board system specified by its system identifier.
AlazarNumOfSystems	Get the number of board systems in a PC.
AlazarOEMDownLoadFPGA	Download an FPGA image file to a board.
AlazarOpen	Open a board handle.
AlazarParseFPGAName	Extract the attributes from an FPGA file name.

4.1.2 Status and information

Name	Purpose
AlazarErrorToText	Convert API error number to NULL terminated string.
AlazarGetBoardKind	Get a board's model from its handle.
AlazarGetBoardRevision	Get the PCB hadware revision level of a digitizer board
AlazarGetChannelInfo	Get the number of bits per sample, and on-board memory size in samples per channel.
AlazarGetCPLDVersion	Get the CPLD version of a board.
AlazarGetDriverVersion	Get the driver version of a board.
AlazarGetParameter	Get a board parameter as a signed 32-bit value.
AlazarGetParameterUL	Get a board parameter as an unsigned 32-bit value.
AlazarGetSDKVersion	Get the API version.
AlazarQueryCapability	Get a board capability as an unsigned 32-bit value.

4.1.3 Configuration and control

Name	Purpose
AlazarConfigureAuxIO	Configure the AUX I/O connector of a board.
AlazarConfigureRecordAverage	Configure a digitizer to add a specified number of records in an accumulator record, and transfer accumulator records rather than ADC sample records.
AlazarConfigureSampleSkipping	Configure the digitizer to sub-sample records in arbitrary, non-uniform intervals.
AlazarCoprocessorDownload	Download an FPGA image to a coprocessor FPGA
AlazarCoprocessorRegisterRead	Read the contents of a coprocessor FPGA register
AlazarCoprocessorRegisterWrite	Write a value to coprocessor a coprocessor FPGA register
AlazarInputControl	Configure the range, coupling, and termination of an input channel.
AlazarInputControlEx	Configure the range, coupling and termination of an input channel.
AlazarResetTimeStamp	Control record timestamp counter resets.
AlazarSetBWLimit	Enable or disable the 20 MHz low-pass filter of an input channel of board.
AlazarSetCaptureClock	Configure the timebase of a board.
AlazarSetClockSwitchOver	Configure the dummy clock.
AlazarSetExternalClockLevel	Set the external clock comparator level of a board.
AlazarSetExternalTrigger	Configure the TRIG IN connector of a board.
AlazarSetLED	Control the LED on the PCI/PCIe mounting bracket of a board.
AlazarSetParameter	Set a board property as a signed 32-bit value.
AlazarSetParameterUL	Set a board property as an unsigned 32-bit value.
AlazarSetTriggerDelay	Specify the amount of time between the arrival of a trigger event, and the acquisition of the first sample of a record.
AlazarSetTriggerOperation	Configure the trigger system of a board.
AlazarSetTriggerOperationForScanning	Configure the trigger system of a board to use an external trigger input when it also uses an external clock signal that does not remain suitable for the board between trigger events.
AlazarSetTriggerTimeOut	Specify the amount of time to wait for a hardware trigger before automatically generating a software trigger event.
AlazarSleepDevice	Turn off the ADC converters.

4.1.4 Acquisition

4.1.4.1 General

Name	Purpose
AlazarForceTrigger	Generate a software trigger event.
AlazarGetStatus	Return a bitmask with acquisition information
AlazarSetRecordSize	Specify the number of samples before and after the sample at the trigger position in a record.
AlazarStartCapture	Arm a board to wait for trigger events.
AlazarTriggered	Determine if a board has received at least one trigger event since the start of an acquisition.
AlazarAllocBufferU16	Allocate a buffer for DMA transfers from a 16-bit digitizer.
AlazarAllocBufferU8	Allocate a buffer for DMA transfers from an 8-bit digitizer.
AlazarFreeBufferU16	Free a buffer allocated by AlazarAllocBufferU16
AlazarFreeBufferU8	Free a buffer allocated by AlazarAllocBufferU8

4.1.4.2 Single-port acquisitions

Name	Purpose
AlazarAbortCapture	Abort a single-ported acquisition to on-board memory.
AlazarBusy	Determine if an acquisition to on-board memory is in progress.
AlazarGetMaxRecordsCapable	Find the maximum number of records that can be captured to on-board memory given a number of samples per record
AlazarGetTriggerAddress	Get the trigger address and timestamp of a record acquired to on-board memory.
AlazarGetTriggerTimestamp	Retrieve the trigger timestamp of a record acquired to on-board memory.
AlazarGetWhoTriggeredBySystemHandle	Get the event that caused a board system, specified by the handle to its master board, to trigger.
AlazarGetWhoTriggeredBySystemID	Get the event that caused a board system, specified by its system identifier, to trigger.
AlazarHyperDisp	Enable the on-board FPGA to divide a record acquired to on-board memory into intervals, and return the minimum and maximum sample values over each interval.
AlazarRead	Transfer all or part of a record acquired to on-board memory.
AlazarReadEx	Transfer all or part of a record acquired to on-board memory when the record has 2,147,483,648 or more samples.

AlazarSetRecordCount	Specify the number of records to capture to on-board memory.

4.1.4.3 Dual-port Asynchronous AutoDMA acquisitions

Name	Purpose
AlazarAbortAsyncRead	Abort an asynchronous AutoDMA acquisition, and release any resources allocated during the acquisition.
AlazarAsyncRead	Add a buffer to the end of a list of buffers available to be filled by a board.
AlazarBeforeAsyncRead	Configure a board to make an asynchronous AutoDMA acquisition.
AlazarCreateStreamFile	Create a binary data file to store sample data for a board.
AlazarForceTriggerEnable	Generate a software trigger enable event.
AlazarPostAsyncBuffer	Add a buffer to the end of a list of buffers available to be filled by a board.
AlazarWaitAsyncBufferComplete	Wait a specified amount of time for a board to receive sufficient trigger events to fill the specified AutoDMA buffer.
AlazarWaitNextAsyncBufferComplete	Wait a specified amount of time for a board to receive sufficient trigger events to fill an AutoDMA buffer managed by the API.

4.1.4.4 Dual-port Synchronous AutoDMA acquisitions

Name	Purpose
AlazarAbortAutoDma	Abort synchronous AutoDMA acquisition
AlazarCloseAUTODma	Release any resources allocated during a synchronous AutoDMA acquisition.
AlazarEvents	Enable a board to wait for the end of an AutoDMA transfer.
AlazarForceTriggerEnable	Generate a software trigger enable event.
AlazarFlushAutoDMA	Stop a synchronous AutoDMA acquisition.
AlazarGetAutoDMAHeaderTimeStamp	Get a record timestamp from an AutoDMA buffer.
AlazarGetAutoDMAHeaderValue	Get an attribute from the record header of an AutoDMA buffer.
AlazarGetAutoDMAPtr	Get a pointer to the header or data portions of a record in an AutoDMA buffer.
AlazarGetNextAutoDMABuffer	Poll for an AutoDMA transfer to complete.
AlazarGetNextBuffer	Poll for an AutoDMA transfer to complete.
AlazarStartAutoDMA	Configure a board to make a synchronous AutoDMA acquisition.

AlazarStopAutoDMA	Inhibit the software from issuing any new DMA request to the board.
AlazarWaitForBufferReady	Wait for an AutoDMA transfer to complete.

4.1.4.5 AlazarDSP Functions

Name	Purpose
AlazarDSPAbortCapture	Abort DSP acquisition
AlazarDSPGetBuffer	Wait for a DSP acquisition's buffer to be filled
AlazarDSPGetModules	List on-FPGA DSP modules
AlazarDSPGetInfo	Get Information about DSP modules
AlazarDSPGenerateWindowFunction	Create a Window function
AlazarFFTSetup	Prepares an FFT acquisition

4.1.4.6 Function Reference

This section provides an alphabetical list of the functions exported by the ATS-SDK, and their descriptions.

4.1.5 AlazarAbortAsyncRead

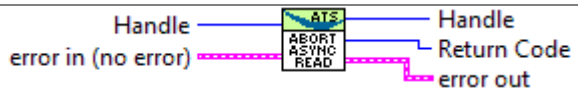
Aborts any in-progress DMA transfers, and cancel any pending transfers.

Syntax

C/C++

```
RETURN_CODE
AlazarAbortAsyncRead (
      HANDLE BoardHandle,
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

Return value

If the function succeeds, it returns ApiSuccess (512).

If the function fails because it was unable to abort an in-progress DMA transfer, it returns ApiDmaInProgress (518).

If **AlazarAbortAsyncRead** fails under Windows because the Windows CancelIo system call failed, the function returns ApiFailed (513). Call the Windows GetLastError API for more information.

If the function fails for some other reason, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

If you have called AlazarAsyncRead or AlazarPostAsyncBuffer, and there are buffers pending, you *must* call **AlazarAbortAsyncRead** before your application exits.

If you do not, when you program exits Microsoft Windows may stop with a blue screen error number 0x000000CB (DRIVER_LEFT_LOCKED_PAGES_IN_PROCESS). Linux may leak the memory used by the DMA buffers.

See Also

AlazarAsyncRead
AlazarPostAsyncBuffer
Using asynchronous AutoDMA

4.1.6 AlazarAbortAutoDma

This function is deprecated. Do not use in new designs.

This routine is used to terminate the AutoDMA capture in cases where the trigger system stopped generating triggers before the buffer was filled by the AutoDMA engine. The routine will populate the buffer with the appropriate number of records that have been successfully captured.

Syntax

```
C/C++
RETURN_CODE
AlazarAbortAutoDMA(
      HANDLE h,
      void* Buffer,
      AUTODMA_STATUS* error,
      U32 r1,
      U32 r2,
      U32 *r3,
      U32 *r4
);
```

Parameters

h

[in] Board identification handle.

Buffer

[out] This Buffer is used to transfer a set of Records from the Device back to the user application.

Error

[out] Error return code.

This error code may be one of the following values.

Identifier	Value	Meaning
ADMA_Completed	0	No errors occurred
ADMA_Success	0	No errors occurred
ADMA_Buffer1Invalid	1	Buffer1 is not a suitable buffer
ADMA_Buffer2Invalid	2	Buffer2 is not a suitable buffer
ADMA_BoardHandleInvalid	3	Board handle is not valid
ADMA_InternalBuffer1Invalid	4	The routine cannot allocate enough memory because system resources are low
ADMA_InternalBuffer2Invalid	5	The routine cannot allocate enough memory because system resources are low
ADMA_OverFlow	6	A hardware overflow occurred

ADMA_InvalidChannel	7	The channel selected is invalid
ADMA_DMAInProgress	8	A memory transfer is in progress
ADMA_UseHeaderNotSet	9	UseHeader must be set
ADMA_HeaderNotValid	10	An invalid header was encountered
ADMA_InvalidRecsPerBuffer	11	RecordCount must be a perfect multiple of RecsPerBuffer

r1

[in] RESERVED.

r2

[in] RESERVED.

r3

[out] RESERVED.

r4

[out] RESERVED.

Return values

See Table 1 for a list of error codes.

Remarks

See Also

AlazarStartAutoDMA
AlazarCloseAUTODma
Using synchronous AutoDMA

4.1.7 AlazarAbortCapture

Abort an acquisition to on-board memory.

Syntax

C/C++

```
RETURN_CODE
AlazarAbortCapture (
      HANDLE BoardHandle,
);
```

LabVIEW

Parameters

BoardHandle
[in] Handle to board.

Return value

The function returns ApiSuccess (512) if it was successful.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

None

See Also

AlazarRead
AlazarHyperDisp
Acquiring to on-board memory

4.1.8 AlazarAllocBufferU16

Allocate a buffer for DMA transfers from a 16-bit digitizer.

Syntax

```
C/C++
U16*
AlazarAllocBufferU16 (
      HANDLE BoardHandle,
      U32    SampleCount
);
```

Parameters

BoardHandle
[in] Handle to board.

SampleCount
[in] Buffer size in samples.

Return value

If it succees, this function returns the base address of a page aligned region in the virtual address space of the calling process.

If it fails, this functions returns NULL.

Remarks

This function returns a buffer whose base address meets the alignment requirement for single- or dual-port DMA transfers of all 16-bit digitizers.

The buffer should be freed by calling AlazarFreeBufferU16.

See Also

AlazarAllocBufferU8
AlazarFreeBufferU16
AlazarFreeBufferU8

4.1.9 AlazarAllocBufferU8

Allocate buffer for DMA transfers from an 8-bit digitizer.

Syntax

```
C/C++
U8*
AlazarAllocBufferU8 (
      HANDLE BoardHandle,
      U32    SampleCount
);
```

Parameters

BoardHandle
[in] Handle to board.

SampleCount
[in] Buffer size in samples.

Return value

If it succees, this function returns the base address of a page aligned region in the virtual address space of the calling process.

If it fails, this functions returns NULL.

Remarks

This function returns a buffer whose base address meets alignment requirements for DMA transfers in single- or dual-port mode from all 8-bit digitizers.

The buffer must be freed by calling AlazarFreeBufferU8.

See Also

AlazarAllocBufferU16
AlazarFreeBufferU16
AlazarFreeBufferU8

4.1.10 AlazarAsyncRead

Add a buffer to the end of a list of buffers available to be filled by the board. When the board receives sufficient trigger events to fill the buffer, the event in the OVERLAPPED will be set to the signaled state.

Syntax

```
C/C++
RETURN_CODE
AlazarAsyncRead(
      HANDLE BoardHandle,
      void *Buffer,
      U32 BytesToRead,
      OVERLAPPED *Overlapped
);
```

Parameters

BoardHandle

[in] Handle to board.

Buffer

[out] Pointer to a buffer to receive sample data from the digitizer board.

BytesToRead

[in] Specifies the number of bytes to read from the board.

Overlapped

[in] Pointer to an OVERLAPPED structure.

The event in the OVERLAPPED structure is set to the signaled state when the read operation completes.

Return value

If the function succeeds in adding the buffer to end of the list of buffers available to be filled by the board, it returns ApiDmaPending (585). When the board fills the buffer, the event in the OVERLAPPED structure is to the signaled state.

If the function fails because the board overflowed its on-board memory, it returns ApiBufferOverflow (582). The board may overflow its on-board memory because the rate at which it is acquiring data is faster than the rate at which it is transferring data from on-board memory to host memory. If this is the case, try reducing the sample rate, number of enabled channels, or amount of time spent processing each buffer.

If the function fails because the buffer is too large for the driver or operating system to prepare for scatter-gather DMA transfer, it returns ApiLockAndProbePagesFailed (586). Try reducing the size of each buffer, or reducing the number of buffers queued by the application.

If the function fails for some other reason, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

AlazarAsyncRead is only available under Windows.

You must call AlazarBeforeAsyncRead before calling **AlazarAsyncRead**.

You must call AlazarAbortAsyncRead before your application exits if you have called **AlazarAsyncRead**, and buffers are pending when you wish to exit your application.

The BytesToRead parameter must be equal to the product of the number of bytes per record, the number of records per buffer, and the number of enabled channels. If record headers are enabled, the number of bytes per record must include the size of the record header (16 bytes).

See Also

AlazarAbortAsyncRead
AlazarBeforeAsyncRead
Using asynchronous AutoDMA

4.1.11 AlazarBeforeAsyncRead

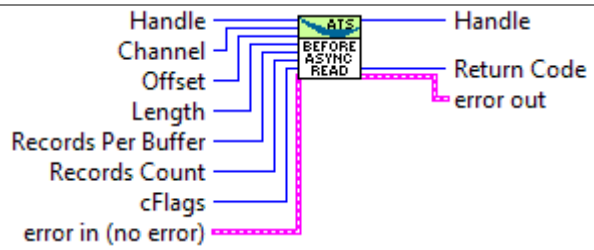
Configure a board to make an asynchronous AutoDMA acquisition.

Syntax

C/C++

```
RETURN_CODE
AlazarBeforeAsyncRead(
   HANDLE BoardHandle,
   U32 ChannelSelect,
   long TransferOffset,
   U32 SamplesPerRecord,
   U32 RecordsPerBuffer,
   U32 RecordsPerAcquisition,
   U32 Flags
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

ChannelSelect

[in] Select the channel to control.

See *Table 4 - Input Channel Configurations* for a list of authorized input channel configurations for each board. The AlazarTech headers define macros CHANNEL_X where X is the name of a channel for all available channels. You can pass these macros directly to this parameter for single-channel acquisitions, or combine them with the bitwise OR operator for multiple channel operation.

TransferOffset

[in] Specify the first sample from each on-board record to transfer from on-board to host memory. This value is a sample relative to the trigger position in an on-board record.

SamplesPerRecord

[in] Specify the number of samples from each record to transfer from on-board to host memory. In DSP-mode, it takes the number of bytes per record instead. See remarks below.

RecordsPerBuffer

[in] Specify the number of records in each buffer. See remarks below.

RecordsPerAcquisition

[in] Specify the number of records in to acquire during acquisition. Set to 0x7fffffff to acquire indefinitely until the acquisition is aborted. Parameter is ignored in Triggered Streaming and Continuous Streaming modes. See remarks below.

Flags

[in] Specify AutoDMA mode and options.

AutoDMA mode must be one of the following values.

Identifier	Meaning
ADMA_TRADITIONAL_MODE (0x00000000)	Acquire multiple records: one per trigger event. Each record may include pre-and post-trigger samples, and a record header that includes its trigger timestamp. If a board has on-board memory and sample interleave is not enabled, each buffer will contain samples organized as follows: R1A, R1B, R2A, R2B … If a board does not have on-board memory, or sample interleave is enabled, the buffer will contain samples organized as follows: R1[AB…], R2[AB…] …
ADMA_NPT (0x00000200)	Acquire multiple records: one per trigger event. Each record contains only post-trigger samples. If a board has on-board memory and sample interleave is not enabled, each buffer will contain samples organized as follows: R1A, R2A, … R1B, R2B … If a board does not have on-board memory, or sample interleave is enabled, the buffer will contain samples organized as follows: R1[AB…], R2[AB…] …
ADMA_CONTINUOUS_MODE (0x00000100)	Acquire a single, gapless record spanning multiple buffers. Do not wait for trigger event before starting the acquisition.

	If a board has on-board memory and sample interleave is not enabled, each buffer will contain samples organized as follows: R1A, R1B. If a board does not have on-board memory, or sample interleave is enabled, the buffer will contain samples organized as follows: R1[AB…]
ADMA_TRIGGERED_STREAMING (0x00000400)	Acquire a single, gapless record spanning multiple buffers. Wait for a trigger event before starting the acquisition. If a board has on-board memory and sample interleave is not enabled, each buffer will contain samples organized as follows: R1A, R1B. If a board does not have on-board memory, or sample interleave is enabled, the buffer will contain samples organized as follows: R1[AB…]

AutoDMA options may be a combination of one or more of the following values.

Identifier	Meaning
ADMA_EXTERNAL_STARTCAPTURE (0x00000001)	If this flag is set, the acquisition will start when the application calls AlazarStartCaputre. If this flag is not set, the acquisition will start before **AlazarBeforeAsyncRead** returns.
ADMA_ENABLE_RECORD_HEADERS (0x00000008)	If this flag is set, precede each record in each buffer with a 16-byte header that includes the record's trigger timestamp. Note that this flag can only be used in "traditional" AutoDMA mode. Record headers are not available in NPT, streaming, or triggered streaming modes.
ADMA_ALLOC_BUFFERS (0x00000020)	If this flag is set, the API will allocate and manage a list of DMA buffers.

	This flag may be used by LabVIEW, and in other high-level development environments, where it may be more convenient for the application to let the API manage a list of DMA buffers, and to receive a copy of data in an application buffer. When this flag is set, the application must call AlazarWaitNextAsyncBufferComplete to wait for a buffer to complete and receive a copy of the data. The application can specify the number of DMA buffers for the API to allocate by calling AlazarSetParameter with the parameter SETGET_ASYNC_BUFFCOUNT before calling **AlazarBeforeAsyncRead**.
ADMA_FIFO_ONLY_STREAMING (0x00000800)	Enable the board to data from its on-FPGA FIFO rather than from on-board memory. When the flag is set, each buffer contains data organized as follows: R0[ABAB…], R1[ABAB…], R2[ABAB] …. That is, each sample from CH A is followed by a sample from CH B. When this flag is not set, each record in a buffer contains a contiguous array of samples for CH A followed by a contiguous array of samples for CH B, where the record arrangement depends on the acquisition mode. Note that this flag must be set if your board does not have on-board memory. For example, an ATS9462-FIFO requires this flag. Also note that this flag must not be set if your board has on-board memory.

ADMA_INTERLEAVE_SAMPLES (0x00001000)	Enable a board to interleave samples from both digitizer channels in dual-channel acquisition mode. This results in higher data transfer rates on boards that support this option. Note that this flag has no effect in single channel mode, and is supported by only PCIe digitizers (except the ATS9462). When the flag is set, each buffer contains data organized as follows: R0[ABAB…], R1[ABAB…], R2[ABAB] …. That is, each sample from CH A is followed by a sample from CH B. When this flag is not set, each record in a buffer contains a contiguous array of samples for CH A followed by a contiguous array of samples for CH B, where the record arrangement depends on the acquisition mode.
ADMA_GET_PROCESSED_DATA (0x00002000)	Enable the API to process each buffer so that the sample data in a buffer is always arranged as in NPT mode: R0A, R1A, R2A, … RB0, R1B, R2B. If this flag is not set, the data arrangement in a buffer depends on the acquisition mode. LabVIEW and other higher-level applications may use this flag to simplify data processing since all data buffers will have the same arrangement independent of the acquisition mode. Note that the ADMA_ALLOC_BUFFERS flag must also be set to use this option.
ADMA_DSP (0x00004000)	Activates the DSP mode that must be used for using the on-FPGA DSP modules such as the on-FPGA FFT.

Return value

If the function succeeds, it returns ApiSuccess (512).

If the function fails, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

The **SamplesPerRecord** parameter must meet certain criterias. Please see Buffer size and alignment for more information. In addition, when the ADMA_DSP is active in admaFlags, this parameter must be given the number of **bytes per record**.

The **RecordsPerBuffer** parameter must be set to 1 in continuous streaming, and triggered streaming AutoDMA modes.

The **RecordsPerAcquisition** parameter must be either:

- An multiple of the **RecordsPerBuffer** parameter, or
- 0x7FFFFFFF to indicate that the acquisition should continue indefinitely until aborted.

This parameter is ignored in Continuous Streaming and Triggered Streaming modes. Instead, the acquisition runs continuously until AlazarAbortAsyncRead is called.

When record headers are not enabled, the total number of bytes per AutoDMA buffer is given by:

```
BytesPerBuffer = BytesPerSample * SamplesPerRecord *
                 RecordsPerBuffer;
```

When record headers are enabled, the total number bytes per AutoDMA buffer is given by:

```
// Each record header occupies 16-bytes
BytesPerBuffer = (16 + BytesPerSample * SamplesPerRecord) *
                 RecordsPerBuffer;
```

For best performance, AutoDMA parameters should be selected so that the total number of bytes per buffer is greater than about 1MB. This allows for relatively long DMA transfers times compared to the time required to prepare a buffer for a DMA transfer, and to re-arm a DMA engine.

ATS460, ATS660, and ATS860 digitizer boards require that AutoDMA parameters be selected so that the total number of bytes per buffer is less than 4MB. Other boards require that the total number of bytes per buffer be less than 64MB.

See Also

AlazarAsyncRead
AlazarAbortAsyncRead
AlazarPostAsyncBuffer
AlazarWaitAsyncBufferComplete
AlazarWaitNextAsyncBufferComplete
Using asynchronous AutoDMA

4.1.12 AlazarAutoCalibrate

Perform a board specific calibration.

Syntax

```
C/C++
RETURN_CODE
AlazarAutoCalibrate(
      HANDLE BoardHandle,
);
```

Parameters

BoardHandle
[in] Handle to board.

Return value

The function returns ApiSuccess (512) if it was successful.

Otherwise, the function returns an error code that should indicate the reason that it failed. See Table 1 for a list of error codes.

Remarks

This function is not implemented.

See Also

4.1.13 AlazarBoardsFound

Determine the number of digitizer boards that were detected in all board systems.

Syntax

```
C/C++
U32 AlazarBoardsFound ();
```

LabVIEW

Parameters

None

Return value

The total number of digitizer boards detected.

Remarks

None

See Also

AlazarNumOfSystems

4.1.14 AlazarBoardsInSystemByHandle

Return the number of digitizer boards in a board system specified by the handle of its master board.

Syntax

C/C++

```
U32 AlazarBoardsInSystemByHandle (
      HANDLE BoardHandle
);
```

LabVIEW

Parameters

BoardHandle

[in] Handle to master board.

Return value

The number of boards in the specified board system.

Remarks

If this function is called with the handle to the master board is a master-slave system, it returns the total number of boards in the system, including the master.

If this function is called with the handle an independent board, it returns 1.

If this function is called with the handle to a slave board in a master-slave system, or with an invalid handle, it returns 0.

See Also

AlazarBoardsInSystemBySystemID
AlazarGetSystemHandle

4.1.15 AlazarBoardsInSystemBySystemID

Return the number of digitizer boards in a board system specified its system ID.

Syntax

C/C++

```
U32 AlazarBoardsInSystemBySystemID(
      U32 SystemId
);
```

LabVIEW

Parameters

SystemId
[in] Board system identifier.

Return value

The number of boards in the specified system.

Remarks

If this function is called with the identifier of a master-slave system, it returns the total number of boards in the system, including the master.

If this function is called with identifier of an independent board system, it returns 1.

If this function is called with the identifier of an invalid board system, it returns 0.

See Also

AlazarBoardsInSystemByHandle
AlazarGetSystemHandle

4.1.16 AlazarBusy

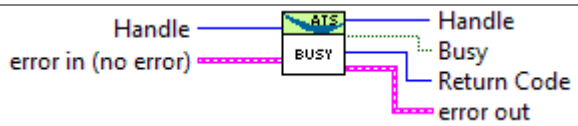
Determine if an acquisition to on-board memory is in progress.

Syntax

C/C++

```
U32 AlazarBusy (
      HANDLE BoardHandle
);
```

LabVIEW



Parameters

BoardHandle
[in] Handle to board.

Return value

If the board is busy acquiring to on-board memory, this function returns 1.

Otherwise, this function returns 0.

Remarks

This function is part of the single-port acquisition API. Once an acquisition to on-board memory is finished, use the AlazarRead, AlazarReadEx, or AlazarHyperDisp functions to transfer sample data from on-board to host memory.

See Also

AlazarHyperDisp
AlazarRead
AlazarReadEx
AlazarStartCapture

© 2003-2015 Alazar Technologies Inc. Version 7.1.1

4.1.17 AlazarClose

This function is obsolete. Do not use in new designs.

Close a board handle.

Syntax

```
C/C++
void AlazarClose (
      HANDLE BoardHandle
);
```

Parameters

BoardHandle
[in] Handle to board.

Return value

If the board is acquiring to on-board memory, this function returns 1. Otherwise, this function returns 0.

Remarks

The API manages board handles internally. This function should only be used in applications that are written for single board digitizer systems.

See Also

AlazarOpen

4.1.18 AlazarCloseAUTODma

This function is deprecated. Do not use in new designs.

This routine will close the AUTODMA capabilities of the device. Only call this upon exit or error.

Syntax

```
C/C++
RETURN_CODE
AlazarCloseAUTODma(
      HANDLE h,
);
```

Parameters

h

[in] Board identification handle.

Return values

See Table 1 for a list of error codes.

Remarks

See Also

AlazarAbortAutoDma
Using synchronous AutoDMA

4.1.19 AlazarConfigureAuxIO

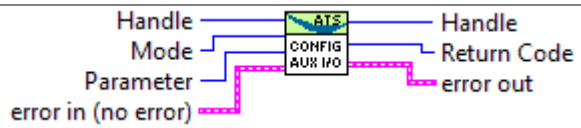
Configure the AUX I/O connector as an input or output signal.

Syntax

C/C++

```
RETURN_CODE
AlazarConfigureAuxIO(
      HANDLE BoardHandle,
      U32 Mode,
      U32 Parameter
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

Mode

[in] Specify AUX I/O mode.

The mode may be one of the following values. See AlazarApi.h for a complete list of list of identifiers.

Identifier	Value	Description
AUX_OUT_TRIGGER	0	Output a trigger signal synchronized with the sample clock.
AUX_IN_TRIGGER_ENABLE	1	Use the edge of a pulse to the AUX I/O connector as an AutoDMA trigger enable signal.
AUX_OUT_PACER	2	Output sample clock divided by user-defined value.
AUX_IN_AUXILIARY	13	Configure the AUX connector as a digital input. Call AlazarGetParameter(handle, 0, GET_AUX_INPUT_LEVEL, &level) to read the digital input level.
AUX_OUT_SERIAL_DATA	14	Use the AUX I/O connector as a general purpose digital output.

If an AUX I/O output mode is selected, then the parameter may be OR'ed with AUX_OUT_TRIGGER_ENABLE (16) to enable the board to use software trigger enable. When this flag is set, the board will: wait for software to call

AlazarForceTriggerEnable to generate a trigger enable event; then wait for sufficient trigger events to capture the records in an AutoDMA buffer; then wait for the next trigger enable event and repeat.

For example, the mode "AUX_OUT_TRIGGER_ENABLE | AUX_OUT_TRIGGER" enables trigger enable, and configures the AUX I/O connector as a trigger output, while "AUX_OUT_TRIGGER_ENABLE | AUX_OUT_PACER" enables trigger enable, and configures the AUX I/O connector as a pacer output.

Parameter

[in] Parameter value.

The meaning of the parameter value depends on the AUX I/O mode.

Mode	Parameter value
AUX_OUT_TRIGGER	The value is ignored
AUX_IN_TRIGGER_ENABLE	The value specifies slope of TTL trigger enable signal: • TRIGGER_SLOPE_POSITIVE (1) The trigger enable signal is the rising edge of a TTL pulse to the AUX I/O connector. • TRIGGER_SLOPE_NEGATIVE (2) The trigger enable signal is the falling edge of a TTL pulse to the AUX I/O connector.
AUX_OUT_PACER	The value specifies sample clock divider. Note that the divider must be greater than 2.
AUX_OUT_SERIAL_DATA	The value specifies the TTL output level: • 0 = TTL low-level • 1 = TTL high level

Return value

If the function succeeds, it returns ApiSuccess (512).

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

The AUX I/O connector generates TTL level signals when configured as an output, and expects TTL level signals when configured as an input.

AUX I/O output signals may be limited by the bandwidth of the AUX output drivers.

The ATS9440 has two AUX I/O connectors: AUX I/O 1 and AUX I/O 2. AUX I/O 1 is configured by firmware as a trigger output signal, while AUX I/O 2 is configured by software using AlazarConfigureAuxIO. A firmware update is required to change the operation of AUX I/O 1.

The ATS9625 and ATS9626 have to AUX I/O connectors: AUX 1 and AUX 2. AUX 1 is configured by software using AlazarConfigureAuxIO, while AUX 2 is configured by default as a trigger output signal. A custom user-programmable FPGA can control the operation of AUX 2 as required by the FPGA designer.

See Also

4.1.20 AlazarConfigureLSB

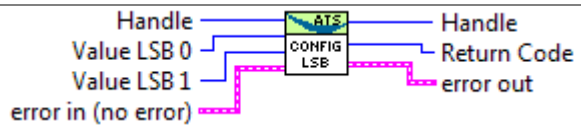
12- and 14-bit digitzers return 16-bit sample values with sample codes stored in the most-significant bits of each 16-bit sample value. By default, the least-significant bits of each sample value are zero filled. Use this option to used these otherwise unused bits as digital outputs.

Syntax

C/C++

```
RETURN_CODE
AlazarConfigureLSB(
      HANDLE BoardHandle,
      U32 ValueLsb0,
      U32 ValueLsb1
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

ValueLsb0
ValueLsb1

[in] Specify the signal to output the least significant bit of each sample value.

The signal may be one of the following values.

Identifier	Value	Description
LSB_DEFAULT	0	Output 0 on the selected bit of each sample.
LSB_EXT_TRIG	1	Output the external trigger input digital state on the selected bit of each sample.
LSB_AUX_IN_0	2	Output the AUX IN 1 digital state on the selected bit of each sample.
LSB_AUX_IN_1	3	Output the AUX IN 2 digatal level on the selected bit of each sample.

Return value

If the function succeeds, it returns ApiSuccess (512).

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

The Configure LSB feature is not available for all boards. Please consult *Table 9 – Miscellaneous Features Support* for a list of supported devices.

The ATS9440 has two AUX I/O connectors: AUX I/O 1 and AUX I/O 2. AUX I/O 1 is configured by firmware as a trigger output signal, while AUX I/O 2 is configured by software using AlazarConfigureAuxIO. A firmware update is required to change the operation of AUX I/O 1.

The ATS9625 and ATS9626 have to AUX I/O connectors: AUX 1 and AUX 2. AUX 1 is configured by software using AlazarConfigureAuxIO, while AUX 2 is configured by default as a trigger output signal. A custom user-programmable FPGA can control the operation of AUX 2 as required by the FPGA designer.

See Also

4.1.21 AlazarConfigureRecordAverage

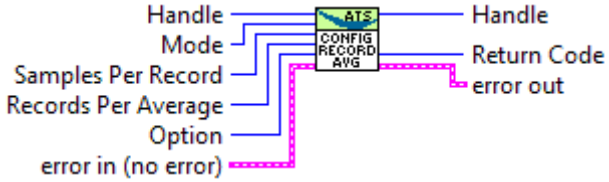
This function configures a digitizer to co-add ADC samples from a specified number of records in an accumulator record, and transfer accumulator records rather than the ADC sample values.

Syntax

C/C++

```
RETURN_CODE
AlazarConfigureRecordAverage(
      HANDLE BoardHandle,
      U32    Mode,
      U32    SamplesPerRecord,
      U32    RecordsPerAverage,
      U32    Options
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

Mode

[in] Select averaging mode.

The parameter may have one of the following values.

Identifier	Value	Description
CRA_MODE_DISABLE	0	Disable record averaging.
CRA_MODE_ENABLE_FPGA_AVE	1	Enable FPGA record averaging.

SamplePerRecord

[in] Specifiy the number ADC samples per accumulator record.

RecordsPerAverage

[in] Specify the number of records to accumulate per average.

Options

[in] Select averaging options.

The parameter may be one of the following values.

Identifier	Value	Description
CRA_OPTION_UNSIGNED	0	Find the sum of unsigned ADC samples

		codes.
CRA_OPTION_SIGNED	1	Find the sum of signed ADC sample codes.

Return value

If the function succeeds, it returns ApiSuccess (512). If the function fails, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

FPGA record averaging is currently supported on the following digitizers:

Model	FPGA version	Driver version
ATS9870	180.0	5.9.8
AXI9870	180.0	5.9.23

When FPGA record averaging is enabled, the digitizer transfers one accumulator record to host memory after "records per average" trigger events have been captured.

Each accumulator record has interleaved samples from CH and CH B:

A1, B1, A2, B2 … An, Bn, where n = "samples per record".

FPGA accmulators are 32-bits wide, so each accumulator value occupies 4-bytes in a buffer. The digitizer transfers multi-byte values in "little endian" byte order.

CH A and CB B accumulator record samples are always transferred to host memory. As a result, the number of bytes per accumulator record is: (samples per record per channel) * (2 channels) * (4 bytes per accumulator sample).

The maximum "records per average" for 8-bit digitizer is $2^{24} - 1 = 16{,}777{,}215$.

Note that "records per average" does not have to be equal to "records per buffer" in AutoDMA mode. The maximum "records per average" is 16,777,215 for 8-bit digitizers, while the maximum "records per buffer" is 8191.

See Also

4.1.22 AlazarConfigureSampleSkipping

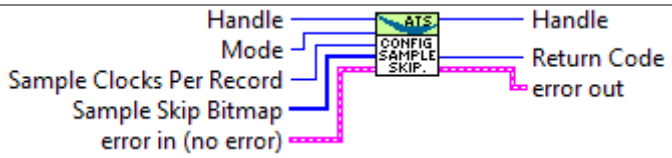
Use this function to allow the digitizer to sub-sample post trigger data in arbitrary, non-uniform intervals. The application specifies which sample clock edges after a trigger event that the digitizer should use to generate sample points, and which sample clock edges that the digitizer should ignore.

Syntax

C/C++

```
RETURN_CODE
AlazarConfigureSampleSkipping(
      HANDLE BoardHandle,
      U32 Mode,
      U32 SampleClocksPerRecord,
      U16 *pSampleSkipBitmap
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

Mode

[in] Select the data skipping mode.

The mode may be one of the following values.

Identifier	Value	Description
SSM_DISABLE	0	Disable sample skipping.
SSM_ENABLE	1	Enable sample skipping.

SampleClocksPerRecord

[in] Specifiy the number of sample clocks per record. This value cannot exceed 65536.

pSampleSkipBitmap

[in] An array of bits that specifies which sample clocks edges should be used to capture a sample point (value = 1), and which sample clock edges should be ignored (bit value = 0).

Return value

If the function succeeds, it returns ApiSuccess (512). If the function fails, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

To enable data skipping, first create a bitmap in memonry that specifies which sample clock edges should generate a sample point, and which sample clock edges should be ingored.

- 1's in the bitmap specify the clock edges that should generate a sample point.
- The total number of 1's in the bitmap must be equal to the number of post-trigger samples per record specified in the call to AlazarSetRecordSize.
- 0's in the bitmap specify the clock edges that should not be used to generate a sample point.
- The total total number of bits in the bitmap is equal to the number of sample clocks in one record.

For example, to receive 16 samples from 32 sample clocks where every other sample clock is ignored, create a bitmap of 32 bits with values { 1 0 1 0 1 0 … 1 0 }, or { 0x5555, 0x5555 }. Note that 16 of the 32 bits are 1's.

And to receive 24 samples from 96 sample clocks where data from every 3 of 4 samples clocks is ignored, create a bitmap of 96 bits with values { 1 0 0 0 1 0 0 0 1 0 0 0 … 1 0 0 0 }, or in { 0x1111, 0x1111, 0x1111, 0x1111, 0x1111, 0x1111 }. Note that 24 of the 96 bits are 1's.

After creating a bitmap, call **AlazarConfigureSampleSkipping** with:
- *Mode* equal to SSM_ENABLE (1).
- *SampleClocksPerRecord* equal to the total number of sample clocks per record.
- *pSampleSkipBitmap* with the address of the U16 array.

To disable data skipping, call **AlazarConfigureSampleSkipping** with Mode equal to SSM_DISABLE (0). The *SampleClocksPerRecord* and *pSampleSkipBitmap* parameters are ignored.

Note that data skipping currently is supported by the ATS9440 with FPGA version 3.1 or later, and only works with post-trigger data data acquired at 125 MSPS or 100 MSPS.

See Also

4.1.23 AlazarCoprocessorDownload

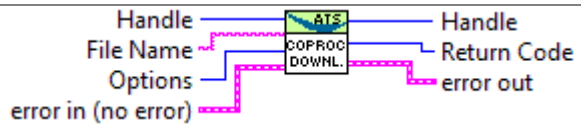
Download an FPGA image in RBF (raw binary file) format to the coprocessor FPGA.

Syntax

C/C++

```
RETURN_CODE
AlazarCoprocessorDownloadA (
      HANDLE BoardHandle,
      char*  FileName
      U32    Options
);

RETURN_CODE
AlazarCoprocessorDownloadW (
      HANDLE BoardHandle,
      WCHAR* FileName
      U32    Options
);
```

LabVIEW



Parameters

BoardHandle
[in] Handle to board.

char, WCHAR**
[in] Path to RBF file.

Options
[in] Download options.

Identifier	Meaning
CPF_OPTION_DMA_DOWNLOAD 0x00000001	Use DMA to transfer the file contents to the coprocessor FPGA.

Return value

The function returns ApiSuccess (512) if the function succeeds. Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

The coprocessor FPGA file must be in RBF (raw binary file) format.

Use AlazarCoprocessorDownloadA to download files with ASCII file paths, and AlazarCoprocessorDownloadW to download files with UNICODE file paths.

Use the CPF_OPTION_DMA_DOWNLOAD option to minimize coprocessor FPGA download times.

See Also

AlazarCoprocessorRegisterRead
AlazarCoprocessorRegisterWrite

4.1.24 AlazarCoprocessorRegisterRead

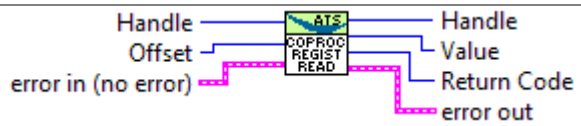
Read the contents of a user-programmable coprocessor FPGA register.

Syntax

C/C++

```
RETURN_CODE
AlazarCoprocessorRegisterRead (
      HANDLE BoardHandle,
      U32 Offset
      U32 *pValue
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

Offset

[in] Register offset.

pValue

[out] Adddress of variable to receive register value.

Return value

The function returns ApiSuccess (512) if the function succeeds. Otherwise, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

None

See Also

AlazarCoprocessorRegisterWrite
AlazarCoprocessorDownload

4.1.25 AlazarCoprocessorRegisterWrite

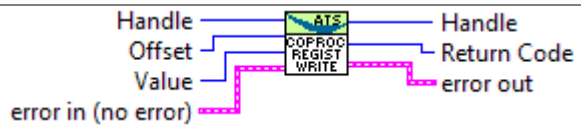
Write a value to a user-programmable coprocessor FPGA register.

Syntax

C/C++

```
RETURN_CODE
AlazarCoprocessorRegisterWrite (
      HANDLE BoardHandle,
      U32 Offset
      U32 Value
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

Offset

[in] Register offset.

Value

[in] Value to write.

Return value

The function returns ApiSuccess (512) if it was able to write to the specified coprocessor FPGA register.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

None

See Also

AlazarCoprocessorRegisterRead
AlazarCoprocessorDownload

4.1.26 AlazarCreateStreamFile

Create a binary data file for this board, and enable saving AutoDMA data from this board to disk.

Syntax

C/C++

```
RETURN_CODE
AlazarCreateStreamFileA(
   HANDLE  BoardHandle,
   char *FilePath
   );

RETURN_CODE
AlazarCreateStreamFileW(
   HANDLE  BoardHandle,
   WCHAR* FilePath
   );
```

LabVIEW

Parameters

BoardHandle

[in] Handle to board.

FilePath

[in] Pointer to a NULL terminated string that specifies the name of the file.

Return values

If this function succeeds, it returns ApiSuccess (512).

If this function fails, it returns ApiFailed (513). Call the Windows GetLastError API for more information.

Remarks

AlazarCreateStreamFileA accepts 8-bit ACSII or MBCS paths, and **AlazarCreateStreamFileW** accepts 16-bit UNICODE paths.

C/C++ applications may use **AlazarCreateStreamFile**. It is defined in AlazarApi.h as follows:

```
#ifdef UNICODE
#define AlazarCreateStreamFile AlazarCreateStreamFileW
#else
#define AlazarCreateStreamFile AlazarCreateStreamFileA
```

If possible, select AlazarBeforeAsyncRead parameters that result in DMA buffers whose length in bytes is evenly divisible into sectors on the volume specified by FilePath. If the DMA buffer length is evenly divisible into sectors, **AlazarCreateStreamFile** disables file caching to obtain the highest possible sequential write performance.

An AutoDMA buffers is saved to disk when an application calls AlazarWaitNextAsyncBufferComplete. For best performance, set the BytesToCopy parameter in AlazarWaitNextAsyncBufferComplete to zero so that data is written to disk without copying data to the user-supplied buffer.

See Also

AlazarWaitNextAsyncBufferComplete
Using asynchronous AutoDMA

4.1.27 AlazarDSPAbortCapture

Aborts any in-progress DMA transfer, cancels any pending transfers and does DSP-related cleanup**.**
This function should be called instead of AlazarAbortAsyncRead() in a standard acquisition configuration. In addition to handling pending and in-flight DMA transfers, it takes care of some cleanup related to the DSP post-processing.

Syntax

C/C++

```
RETURN_CODE
AlazarDSPAbortCapture(
   HANDLE  BoardHandle
   );
```

LabVIEW

Warning
Whereas it is not necessary to call AlazarAbortAsyncRead() to clean after a standard acquisition, calling AlazarDSPAbortCapture() is strictly required.

Parameters

BoardHandle
[in] Handle to board.

4.1.28 AlazarDSPGetBuffer

Waits until a buffer becomes available or an error occurs.
This function should be called instead of AlazarWaitAsyncBufferComplete() in a standard acquisition configuration.

Syntax

```
C/C++
RETURN_CODE
AlazarDSPGetBuffer(
   HANDLE  BoardHandle,
   void* buffer,
   U32 timeout_ms
   );
```

Parameters

BoardHandle

[in] Handle to board.

buffer

[in, out] Pointer to the DMA buffer we want to retrieve. This must correspond to the first DMS buffer posted to the board that has not yet been retrieved.

timeout_ms

[in] Time to wait for the buffer to be ready before returning with an ApiWaitTimeout error.

4.1.29 AlazarDSPGetModules

Queries the list of DSP modules in a given board. This function allows to query the list of DSP modules for a digitizer board. modules is a pointer to an array of DSP modules to be filled by this function. The numEntries parameter specifies how many modules can be added by the function to the modules array. Lastly, the numModules array specifies how many modules are avaiable on the specified board.

modules can be NULL. In this case, the only interest of this function is to return the number of modules available. Please note that numEntries must be zero if modules is NULL.

numModules can be NULL. In this case, it is ignored.

This function is typically called twice. First without a modules array to query the number of available modules, and a second time after allocating an appropriate array.

```
U32 numModules;
U32 retCode = AlazarDSPGetModules(handle, 0, NULL, &numModules);
// Error handling
dsp_module_handle modules[numModules];
retCode = AlazarDSPGetModules(handle, numModules, modules, NULL);
// Error handling
```

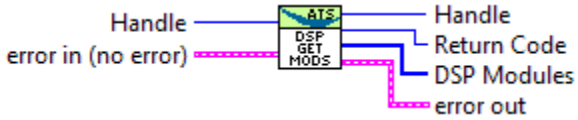
Under LabVIEW, the sequence of two calls to the C function is done by the sub-VI internally. It is only necessary to call the AlazarDSPGetModules sub-VI once.

Syntax

C/C++

```
RETURN_CODE
AlazarDSPGetModules (
   HANDLE  BoardHandle,
   U32 numEntries,
   dsp_module_handle* modules,
   U32* numModules
   );
```

LabVIEW



Parameters

BoardHandle

[in] The handle of the board to query DSP modules for.

numEntries

[in] The maximum number of entries that the function can fill in the modules array. Must be zero if modules is NULL.

modules

[out] The array where this function fills the dsp_module_handle elements. Can be NULL.

numModules

[out] Returns the number of DSP modules available on this board. Ignored if NULL.

4.1.30 AlazarDSPGetNextBuffer

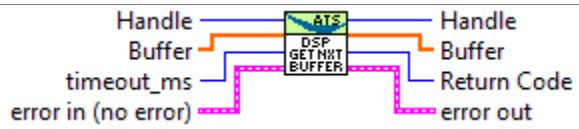
Equivalent of AlazarDSPGetBuffer() to call with ADMA_ALLOC_BUFFERS.
This function should be called instead of AlazarWaitNextAsyncBufferComplete() when ADMA_ALLOC_BUFFERS is used. See the documentation of this function for more information.

Syntax

C/C++

```
RETURN_CODE
AlazarDSPGetNextBuffer(
   HANDLE  BoardHandle,
   void* buffer,
   U32 bytesToCopy,
   U32 timeout_ms
   );
```

LabVIEW



Parameters

BoardHandle
[in] Handle to board.

buffer
[out] Pointer to a buffer to receive sample data from the digitizer board.

bytesToCopy
[in] The number of bytes to copy into the buffer.

timeout_ms
[in] Time to wait for the buffer to be ready before returning with an ApiWaitTimeout error.

4.1.31 AlazarDSPGetInfo

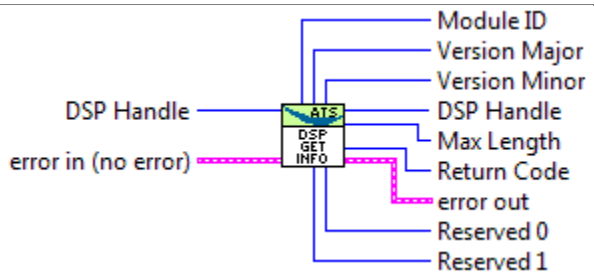
Get information about a specific on-FPGA DSP module. Use this function to query the type of a DSP module, as well as other information.

Syntax

C/C++

```
RETURN_CODE
AlazarDSPGetInfo(
   sp_module_handle  dspHandle,
   U32* dspModuleId,
   U16* versionMajor,
   U16* versionMinor,
   U32* maxLength,
   U32* reserved0,
   U32* reserved1
   );
```

LabVIEW



Parameters

dspHandle

[in] Handle to the DSP module to query.

dspModuleId

[out] The identifier of the DSP module.

Identifier	Meaning
DSP_MODULE_NONE 0x0000FFFF	No DSP module
DSP_MODULE_FFT	FFT module
DSP_MODULE_PCD	

versionMajor

[out] The major version number of the DSP module

versionMinor

[out] The minor version number of the DSP module

maxLength

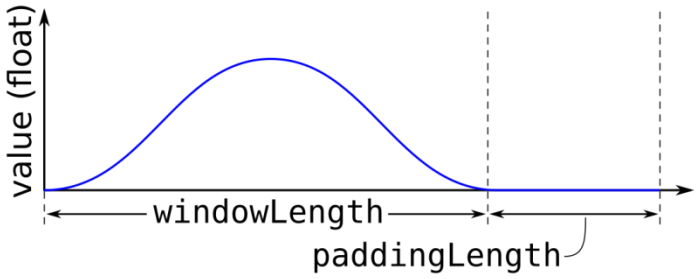
[out] The maximum length of the records that can be processed.

reserved0
Reserved parameter. Ignored.

reserved1
Reserved parameter. Ignored.

4.1.32 AlazarDSPGenerateWindowFunction

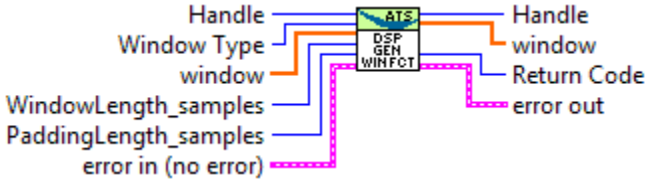
Fills an array with a generated window function and pads it with zeros. See remarks below.



Syntax

```
C/C++
RETURN_CODE
AlazarDSPGenerateWindowFunction(
   U32 windowType,
   float* window,
   U32 windowLength_samples,
   U32 paddingLength_samples
   );
```

LabVIEW



Parameters

windowType

[in] The type of window to generate.

Identifier	Meaning
DSP_WINDOW_NONE = 0x0	No window
DSP_WINDOW_HANNING = 0x1	Hanning window
DSP_WINDOW_HAMMING = 0x2	Hamming window
DSP_WINDOW_BLACKMAN = 0x3	Blackman window
DSP_WINDOW_BLACKMAN_HARRIS = 0x4	Blackman-Harris window
DSP_WINDOW_BARTLETT = 0x5	Bartlett window

window

[out] Array to be filled with the window function. It must be at least windowLength_samples + paddingLength_samples long.

windowLength_samples

[in] The size of the window to generate.

paddingLength_samples

[in] The number of samples after the window function to pad with zeros

Remarks

Using Python, the window array is not allocated first then passed as an output parameter. Instead, it is directly returned from the function as a newly allocated NumPy array.

AlazarErrorClusterToCode
Extracts the ATS-SDK error code from a LabVIEW error cluster.

Syntax

LabVIEW

Parameters

Error in
[in] LabVIEW error cluster.

Return value

ATS-SDK error code

Remarks

This VI is needed because the error code in the LabVIEW error cluster is different from the ATS-SDK error code. See AlazarErrorCodeToCluster for more information.

See Also

4.1.33 AlazarErrorCodeToCluster

Generates a LabVIEW error cluster based on a error code and an optional message.

Syntax

LabVIEW

Parameters

Error Code

[in] Return code from API function.

Message

[in] Optional string to add to the error cluster internal string.

Return value

LabVIEW error cluster

Remarks

The error code in the output error cluster equals the input error code plus 5000. This is due to the fact that the error code range of the ATS-SDK is revered for LabVIEW's usage. With this shift, the resulting error code falls in a range reserved for custom error codes.

See Also

4.1.34 AlazarErrorToText

Convert a numerical return code to a NULL terminated string.

Syntax

C/C++

```
const char* AlazarErrorToText(RETURN_CODE retCode);
```

LabVIEW

Parameters

retCode

[in] Return code from API function.

Return value

Null terminated or VB string containing the identifier name.

Remarks

It is often easier to work with a descriptive error name rather than an error number. For example:

```
RETURN_CODE retCode = ApiSuccess;
printf("Return code %u means %s.\n"),
   retCode,
   AlazarErrorToText(retCode));
```

The output from this code would be the following:

```
Return code 512 means ApiSuccess.
```

See Also

4.1.35 AlazarEvents

This function is deprecated. Do not use in new designs.

This function allows a user to enable or disable usage of software events in AutoDMA mode. The driver manages the event processing and a user can only use an event in conjunction with the API AlazarWaitForBufferReady (...). When the events are enabled AlazarWaitForBufferReady(...) will wait until an AutoDMA buffer is available to the users application. For a complete understanding of the Usage of the API AlazarEvents (...) refer to the pseudo-code example provided in the API AlazarWaitForBufferReady (...).

Syntax

```
C/C++
RETURN_CODE
AlazarEvents(
      HANDLE h,
      U32 enable
);
```

Parameters

h

[in] Handle to the device.

enable

[in] This parameter may have one of the following values.

Identifier	Value	Description
SW_EVENTS_OFF	0	Disable events usage
SW_EVENTS_ON	1	Enable event usage

Return value

ApiSuccess (512) signifies that the API was able to enable the events

ApiFailed (513) signifies that the current driver does not support this feature

Remarks

This functionality is only present on the ATS460, ATS660 and ATSS860 devices. It must be called before calling AlazarStartAutoDMA().

If AlazarEvents(h,1) was not used, calling AlazarWaitForBuffer(…) will return 672 and will not disrupt any ongoing signal captures.

See Also

AlazarWaitForBufferReady
Using synchronous AutoDMA

4.1.36 AlazarFFTSetup

Configure the board for an FFT acquisition.
This function needs to be called in the board configuration procedure, therefore before AlazarBeforeAsyncRead().

Syntax

C/C++

```
RETURN_CODE AlazarFFTSetup(
   dsp_module_handle dspHandle,
   U16 inputChannelMask,
   U32 recordLength_samples,
   U32 fftLength_samples,
   U32 outputFormat,
   U32 footer,
   U32 reserved,
   U32* bytesPerOutputRecord
);
```

LabVIEW

Parameters

dspHandle
[in] The FFT module to configure.

inputChannelMask
[in] The channels to acquire data from. This must be CHANNEL_A.

recordLenght_samples
[in] The number of points per record to acquire. This needs to meet the usual requirements for the number of samples per record. Please see the documentation of AlazarBeforeAsyncRead() for more information.

fftLength_samples
[in] The number of points per FFT. This value must be:

- A power of two
- Greater than or equal to recordLength_samples
- Less that or equal to the maximum FFT size, as returned by the AlazarDSPGetInfo() function.

outputFormat
[in] Describes what data is output from the FFT post-processing module.

Identifier	Value	Description
FFT_OUTPUT_FORMAT_U32_AMP2	0x0000	U32 type and amplitude squared scale
FFT_OUTPUT_FORMAT_U16_LOG	0x0001	U16 format and log of

		amplitude scale
FFT_OUTPUT_FORMAT_U16_AMP2	0x0101	U16 format and amplitude squared scale
FFT_OUTPUT_FORMAT_U8_LOG	0x2	U8 format and log of amplitude scale
FFT_OUTPUT_FORMAT_U8_AMP2	0x102	U8 format and amplitude squared scale
FFT_OUTPUT_FORMAT_REAL_S32	0x3	S32 format and real-valued output
FFT_OUTPUT_FORMAT_IMAG_S32	0x4	S32 format and imaginary-valued output
FFT_OUTPUT_FORMAT_RAW_PLUS_FFT	0x1000	Can be OR'ed with any of the above to add the signed raw signal to the output

footer

[in] Describes if a footer is attached to the returned records.

Identifier	Value	Description
FFT_FOOTER_NONE	0	No footer
FFT_FOOTER_NPT	1	NPT footer

reserved

Reserved parameter. Ignored.

bytesPerOutputRecord

[out] Returns the number of bytes in each record coming our of the FFT module. This value can be used to know how long the allocated DMA buffers must be.

4.1.37 AlazarFFTSetWindowFunction

Sets the window function to use with an on-FPGA FFT module. Records a window function to an AlazarTech digitizer's memory. This window function will be used during all subsequent acquisitions that use the on-FPGA DSP module. This function should be called before AlazarFFTSetup(). It does not have to be called every time an acquisition is done. It can be located in the board configuration section.

Syntax

C/C++

```
RETURN_CODE AlazarFFTSetWindowFunction(
   dsp_module_handle dspHandle,
   U32 samplesPerRecord,
   float* realWindowArray,
   float* imagWindowArray
);
```

LabVIEW

Parameters

dspHandle

[in] The handle of the FFT DSP module to set the window function for.

samplesPerRecord

[in] The number of samples in the window function array.

realWindowArray

[in] The real window function array. Passing NULL is equivalent to passing an array filled with ones.

imagWindowArray

[in] The imaginary window function array. Passing NULL is equivalent to passing an array filled with zeros.

Warning

Please note that the window function is not compatible with the FFT verification mode.

4.1.38 AlazarFreeBufferU16

Free a buffer allocated with AlazarAllocBufferU16.

Syntax

```
C/C++
RETURN_CODE
AlazarFreeBufferU16 (
      HANDLE BoardHandle,
      U16*    Buffer
);
```

Parameters

BoardHandle
[in] Handle to the device.

Buffer
[in] Base address of buffer.

Return value

If this function succeeds, it returns ApiSuccess (512).

If this function fails, it returns a value other than ApiSuccess (512). See Table 1 for a list of error codes.

Remarks

This function must be called to release a buffer allocated by AlazarAllocBufferU16.

See Also

AlazarAllocBufferU16
AlazarAllocBufferU8
AlazarFreeBufferU8

4.1.39 AlazarFreeBufferU8

Free a buffer allocated with AlazarAllocBufferU8.

Syntax

```
C/C++
RETURN_CODE
AlazarFreeBufferU8(
      HANDLE BoardHandle,
      U8*    Buffer
);
```

Parameters

BoardHandle
[in] Handle to the device.

Buffer
[in] Base address of buffer.

Return value

If this function succeeds, it returns ApiSuccess (512).

If this function fails, it returns the value other than ApiSuccess (512). See Table 1 for a list of error codes.

Remarks

This function must be called to free a buffer allocated with AlazarAllocBufferU8.

See Also

AlazarAllocBufferU16
AlazarAllocBufferU8
AlazarFreeBufferU16

4.1.40 AlazarFlushAutoDMA

This function is deprecated. Do not use in new designs.

The primary use of the API is to stop a Synchronous NPT acquisition. Scanning type applications are usually configured such that the data capture is ongoing and stopping is done by an external event. In this case trigger events have stopped and this API permits the last buffer to be returned to the application.

Syntax

```
C/C++
long AlazarFlushAutoDMA (HANDLE h);
```

Parameters

h

[in] Handle to the device.

Return value

The number of valid triggers in the last buffer.

Remarks

Suppose an acquisition is running and all of the sudden, triggers stop coming in. Once the software has determined that the acquisition is to be aborted, **AlazarFlushAutoDMA** should be called. The routine will automatically generate the missing triggers in order to complete the last buffer.

A last call to AlazarGetNextAutoDMABuffer is needed to read the LAST buffer. You will get ApiFailed as a return value from AlazarGetNextAutoDMABuffer indicating a successful last buffer. At this point, depending on your design, you may terminate the program or start a new acquisition.

NOTE:
Internally, this routine calls AlazarStopAutoDMA so as not to allow the software to re arm any new DMA requests. Only a call to AlazarStartAutoDMA will reset this action.

See Also

AlazarGetNextAutoDMABuffer
AlazarStartAutoDMA
Using synchronous AutoDMA

© 2003-2015 Alazar Technologies Inc. Version 7.1.1

4.1.41 AlazarForceTrigger

Generate a software trigger event.

Syntax

C/C++

```
RETURN_CODE
AlazarForceTrigger (
      HANDLE BoardHandle,
);
```

LabVIEW

Parameters

BoardHandle
[in] Handle to board.

Return value

The function returns ApiSuccess (512) if it was successful.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

None

See Also

AlazarSetExternalTrigger
AlazarSetTriggerDelay
AlazarSetTriggerOperation
AlazarSetTriggerTimeOut

4.1.42 AlazarForceTriggerEnable

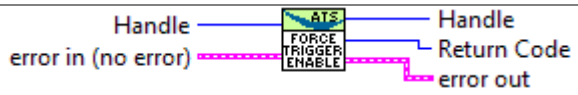
Generate a software trigger enable event.

Syntax

C/C++

```
RETURN_CODE
AlazarForceTriggerEnable (
      HANDLE BoardHandle,
);
```

LabVIEW



Parameters

BoardHandle
[in] Handle to board.

Return value

The function returns ApiSuccess (512) if it was successful.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

If the AUX I/O connector has been configured as a trigger enable input, an application can call this function to generate a software trigger enable event while the board is waiting for hardware to supply an edge to the the AUX input.

If the AUX I/O connector has been configured as a trigger enable output, an application should call this function to generate a trigger enabe event.

See Also

AlazarConfigureAuxIO

4.1.43 AlazarGetAutoDMAHeaderTimeStamp

This function is obsolete. Do not use in new designs.

This routine is a helper function, which can be used to retrieve the 40-bit TimeStamp from the header of a particular record. The resulting number is composed of both the TimeStampHighPart and TimeStampLowPart thus alleviating the user from calculating the time stamp using the header values.

Syntax

```
C/C++
float
AlazarGetAutoDMAHeaderTimeStamp(
      HANDLE h,
      U32 Channel,
      void* DataBuffer,
      U32 Record,
      AUTODMA_STATUS *error
);
```

Parameters

h

[in] Handle to the device.

Channel

[in] This parameter may be one of the following identifiers or values.

Identifier	Value
CHANNEL_A	1
CHANNEL_B	2

DataBuffer

[in] The data buffer as returned from AlazarGetNextAutoDMABuffer.

Record

[in] Signifies the record number of interest for the given Data Buffer.

Error

[out] Error return code.

This error code may be one of the following values.

Identifier	Value	Meaning
ADMA_Completed	0	No errors occurred
ADMA_Success	0	No errors occurred
ADMA_Buffer1Invalid	1	Buffer1 is not a suitable buffer
ADMA_Buffer2Invalid	2	Buffer2 is not a suitable buffer
ADMA_BoardHandleInvalid	3	Board handle is not valid
ADMA_InternalBuffer1Invalid	4	The routine cannot allocate enough

		memory because system resources are low
ADMA_InternalBuffer2Invalid	5	The routine cannot allocate enough memory because system resources are low
ADMA_OverFlow	6	A hardware overflow occurred
ADMA_InvalidChannel	7	The channel selected is invalid
ADMA_DMAInProgress	8	A memory transfer is in progress
ADMA_UseHeaderNotSet	9	UseHeader must be set
ADMA_HeaderNotValid	10	An invalid header was encountered
ADMA_InvalidRecsPerBuffer	11	RecordCount must be a perfect multiple of RecsPerBuffer

Return value

Upon success, i.e. error==ADMA_Success, the TimeStamp will be returned in a floating-point format.

If an error has occurred then 0 will be returned.

Remarks

See Also

AlazarGetAutoDMAHeaderValue
AlazarGetAutoDMAPtr

4.1.44 AlazarGetAutoDMAHeaderValue

This function is deprecated. Do not use in new designs.

This routine is a helper function that can be used to retrieve all the various elements available in the header of an AutoDMA record. It will only operate on records that were captured when the Use Header variable in AlazarStartAutoDMA was set to a 1.

Syntax

```
C/C++
U32
AlazarGetAutoDMAHeaderValue(
      HANDLE h,
      U32 Channel,
      void* DataBuffer,
      U32 Record,
      U32 Parameter,
      AUTODMA_STATUS *error
);
```

Parameters

h

[in] Handle to the device.

Channel

[in] This parameter may be one of the following identifiers or values.

Identifier	Value
CHANNEL_A	1
CHANNEL_B	2

DataBuffer

[in] The data buffer as returned from AlazarGetNextAutoDMABuffer.

Record

[in] Signifies the record number of interest for the provided Data Buffer.

Parameter

[in] Signifies which element the routine should extract from the record's header.

This parameter may be one of the following identifiers or values.

Identifier	Value
ADMA_CLOCKSOURCE	1
ADMA_CLOCKEDGE	2
ADMA_SAMPLERATE	3
ADMA_INPUTRANGE	4
ADMA_INPUTCOUPLING	5
ADMA_IMPUTIMPEDENCE	6

ADMA_EXTTRIGGERED	7
ADMA_CHA_TRIGGERED	8
ADMA_CHB_TRIGGERED	9
ADMA_TIMEOUT	10
ADMA_THISCHANTRIGGERED	11
ADMA_SERIALNUMBER	12
ADMA_SYSTEMNUMBER	13
ADMA_BOARDNUMBER	14
ADMA_WHICHCHANNEL	15
ADMA_SAMPLERESOLUTION	16
ADMA_DATAFORMAT	17

Error

[out] Error return code.

This error code may be one of the following values.

Identifier	Value	Meaning
ADMA_Completed	0	No errors occurred
ADMA_Success	0	No errors occurred
ADMA_Buffer1Invalid	1	Buffer1 is not a suitable buffer
ADMA_Buffer2Invalid	2	Buffer2 is not a suitable buffer
ADMA_BoardHandleInvalid	3	Board handle is not valid
ADMA_InternalBuffer1Invalid	4	The routine cannot allocate enough memory because system resources are low
ADMA_InternalBuffer2Invalid	5	The routine cannot allocate enough memory because system resources are low
ADMA_OverFlow	6	A hardware overflow occurred
ADMA_InvalidChannel	7	The channel selected is invalid
ADMA_DMAInProgress	8	A memory transfer is in progress
ADMA_UseHeaderNotSet	9	UseHeader must be set
ADMA_HeaderNotValid	10	An invalid header was encountered
ADMA_InvalidRecsPerBuffer	11	RecordCount must be a perfect multiple of RecsPerBuffer

Return value

IF error==ADMA_Success, then the value of the asked Parameter is returned.

Remarks

See Also

AlazarGetAutoDMAPtr
AlazarGetAutoDMAHeaderTimeStamp

4.1.45 AlazarGetAutoDMAPtr

This function is deprecated. Do not use in new designs.

This routine is a helper function used to retrieve a pointer to the first data element or first header element of a particular record. If DataOrHeader is set to 1, then the resulting pointer must be cast to PALAZAR_HEADER type. The user can then use the pointer to access any of the header variables.
Ex. PALAZAR_HEADER p = (PALAZAR_HEADER) AlazarGetAutoDMAPtr (…);

Syntax

```
C/C++
void *
AlazarGetAutoDMAPtr(
      HANDLE h,
      U32 DataOrHeader,
      U32 Channel,
      void* DataBuffer,
      U32 Record,
      AUTODMA_STATUS *error
);
```

Parameters

h

[in] Handle to the device.

DataOrHeader

[in] Instruct the routine to return a pointer for the data or header portion.

This parameter may be one of the following values.

Value	Meaning
0	Return the pointer for the data portion.
1	Return the pointer for the header portion.

Channel

[in] This parameter may be one of the following identifiers or values.

Identifier	Value
CHANNEL_A	1
CHANNEL_B	2

DataBuffer

[in] The data buffer as returned from AlazarGetNextAutoDMABuffer.

Record

[in] Signifies the record number of interest for the given Data Buffer.

Error

[out] Error return code.

This error code may be one of the following values.

Identifier	Value	Meaning
ADMA_Completed	0	No errors occurred
ADMA_Success	0	No errors occurred
ADMA_Buffer1Invalid	1	Buffer1 is not a suitable buffer
ADMA_Buffer2Invalid	2	Buffer2 is not a suitable buffer
ADMA_BoardHandleInvalid	3	Board handle is not valid
ADMA_InternalBuffer1Invalid	4	The routine cannot allocate enough memory because system resources are low
ADMA_InternalBuffer2Invalid	5	The routine cannot allocate enough memory because system resources are low
ADMA_OverFlow	6	A hardware overflow occurred
ADMA_InvalidChannel	7	The channel selected is invalid
ADMA_DMAInProgress	8	A memory transfer is in progress
ADMA_UseHeaderNotSet	9	UseHeader must be set
ADMA_HeaderNotValid	10	An invalid header was encountered
ADMA_InvalidRecsPerBuffer	11	RecordCount must be a perfect multiple of RecsPerBuffer

Return value

See Table 1 for a list of error codes.

Remarks

See Also

AlazarGetAutoDMAHeaderTimeStamp
AlazarGetAutoDMAHeaderValue

4.1.46 AlazarGetBoardBySystemHandle

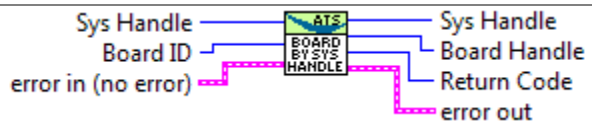
Get a handle to a board in a board system where the board system is specified by a handle to its master board, and the board by its identifier within the board system.

Syntax

C/C++

```
HANDLE
AlazarGetBoardBySystemHandle (
      HANDLE BoardHandle,
      U32 BoardId
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to master board.

BoardId

[in] Board identifier in board system.

Return value

This function returns a handle to the specified board if it was found.

The function returns NULL if the master board handle is invalid, or a board with the specified board identifier was not found in the specified board system.

Remarks

The board identifier of a master board in a board system is always 1.

See Also

AlazarGetBoardBySystemID
AlazarGetSystemHandle

4.1.47 AlazarGetBoardBySystemID

Get a handle to a board in a board system where the board system is specified its system identifier and the board by its board identifier within the board system.

Syntax

C/C++

```
HANDLE
AlazarGetBoardBySystemID (
      U32 SystemId,
      U32 BoardId
);
```

LabVIEW

Parameters

SystemId

[in] System identifier number.

BoardId

[in] Board identifier in system.

Return value

This function returns a handle to the specified board if it was found.

It returns NULL if the board system with the specified ID was not found, or a board with the specified ID was not found within the specified board system.

Remarks

See Also

AlazarGetBoardBySystemHandle
AlazarGetSystemHandle

4.1.48 AlazarGetBoardKind

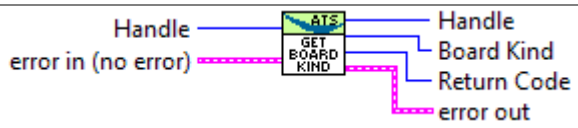
Get a board model identifier of the board associated with a board handle.

Syntax

C/C++

```
U32 AlazarGetBoardKind (
      HANDLE BoardHandle
);
```

LabVIEW



Parameters

BoardHandle
[in] Handle to board.

Return value

If the function succeeds, it returns a non-zero board model identifier.

If the function fails, it returns 0.

Remarks

The following lists currently supported board model identifiers and their values. See AlazarApi.h for a complete list of board type identifiers.

Identifier	Value
ATS850	1
ATS310	2
ATS330	3
ATS460	7
ATS860	8
ATS660	9
ATS9462	11
ATS9870	13
ATS9350	14
ATS9325	15
ATS9440	16
ATS9410	17
ATS9351	18
ATS9850	21
ATS9625	22

© 2003-2015 Alazar Technologies Inc. Version 7.1.1

ATS9626	24
ATS9360	25
AXI9870	26
ATS9370	27
ATU7825	28
ATS9373	29
ATS9416	30

See Also

4.1.49 AlazarGetBoardRevision

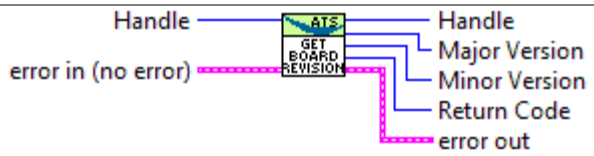
Get the PCB hadware revision level of a digitizer board.

Syntax

C/C++

```
RETURN_CODE
AlazarGetBoardRevision (
      HANDLE boardHandle,
      U8 *MajorVersion,
      U8 *MinorVersion
);
```

LabVIEW



Parameters

BoardHandle
[in] Board handle.

MajorVersion
[out] PCB major version number.

RevisionNumber
[out] PCB minor version number.

Return value

The function returns ApiSuccess (512) if it was successful.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

AlazarTech periodically updates the PCB hadware of its digitizers to improve functionality. Many PCIE digitizers can report the PCB hadware revision to software. Note that this function is not supported on PCI digitizer boards.

See Also

AlazarGetCPLDVersion
AlazarGetSDKVersion

4.1.50 AlazarGetChannelInfo

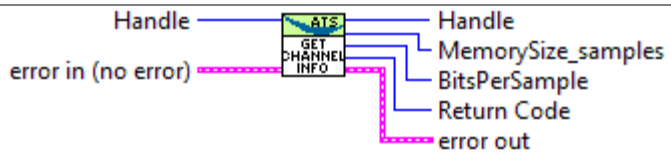
Get the on-board memory in samples per channel, and sample size in bits per sample.

Syntax

C/C++

```
RETURN_CODE
AlazarGetChannelInfo (
      HANDLE BoardHandle,
      U32 *MemorySizeInSamples,
      U8 *BitsPerSample
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

MemorySizeInSamples

[out] The on-board memory size in samples per channel.

bitsPerSample

[out] The number of bits per sample.

Return value

The function returns ApiSuccess (512) if it was successful.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

The ATS9325, ATS9350, ATS9351, ATS9360, ATS9416, ATS9373, ATS9625, ATS9626, ATS9850, AXI9870 and ATS9870 can dedicate all on-board memory to a single channel. The on-board memory size reported by these boards is the maximum samples per channel in single channel mode. In dual-channel mode, the on-board memory is shared equally between both channels.

The ATS9440 with 4-input channels can dedicate all on-board memory to a single channel, two channels, or four channels. The on-board memory size reported by these boards is the maximum samples per channel in single channel mode. In dual-channel mode, the on-board memory is shared equally between two channels. And in four channel

mode, the on-board memory is shared equally between all four channels. The digitizer does not support 3-channel operation.

4.1.51 AlazarGetCPLDVersion

Get the CPLD version number of the specified board.

Syntax

C/C++

```
RETURN_CODE
AlazarGetCPLDVersion(
      HANDLE  BoardHandle,
      U8 *MajorNumber,
      U8 *MinorNumber
);
```

LabVIEW

Parameters

BoardHandle
[in] Handle to board.

MajorNumber
[out] The CPLD major revision number.

MinorNumber
[out] The CPLD minor revision number.

Return value

The function returns ApiSuccess (512) if it was successful.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

See Also

AlazarGetDriverVersion
AlazarGetSDKVersion

4.1.52 AlazarGetDriverVersion

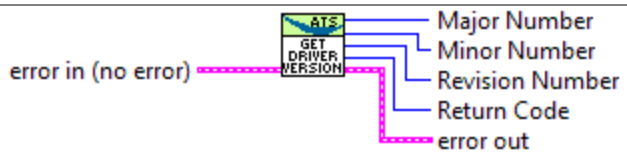
Get the device driver version of the most recently opened device.

Syntax

C/C++

```
RETURN_CODE
AlazarGetDriverVersion (
      U8 *MajorNumber,
      U8 *MinorNumber,
      U8 *RevisionNumber
);
```

LabVIEW



Parameters

MajorNumbr
[out] The driver major version number.

MinorNumber
[out] The driver minor version number.

RevisionNumber
[out] The driver revision number.

Return value

The function returns ApiSuccess (512) if it was successful.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

Driver releases are given a version number with the format X.Y.Z where: X is the major release number, Y is the minor release number, and Z is the minor revision number.

See Also

AlazarGetCPLDVersion
AlazarGetSDKVersion

4.1.53 AlazarGetMaxRecordsCapable

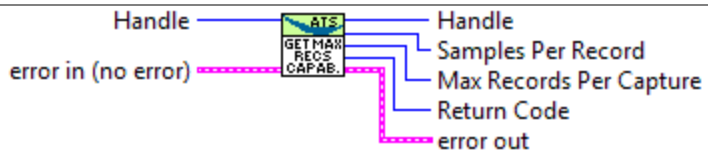
Calculate the maximum number of records that can be captured to on-board memory given the requested number of samples per record.

Syntax

C/C++

```
RETURN_CODE
AlazarGetMaxRecordsCapable (
      HANDLE BoardHandle,
      U32 SamplesPerRecord,
      U32 *MaxRecordsPerCapture
);
```

LabVIEW



Parameters

BoardHandle

[in] The handle a board in a board system.

SamplesPerRecord

[in] The desired number of samples per record.

MaxRecordsPerCapture

[out] The maximum number of records per capture possible with the requested samples per record.

Return value

The function returns ApiSuccess (512) if it was successful.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

This function is part of the single port API. It should not be used with AutoDMA API functions.

See Also

AlazarHyperDisp
AlazarRead
AlazarReadEx

4.1.54 AlazarGetNextAutoDMABuffer

This function is deprecated. Do not use in new designs.

After an application has called AlazarStartAutoDMA the application must call AlazarGetNextAutoDMABuffer to retrieve the data buffers. Because of the nature of Auto Dma, two buffers are required. The device driver dll will arbitrate to which buffer the data will be returned. After a buffer has been filled, variable WhichOne equals the buffer id, thus if the id is 0 then Buffer1 was used and likewise if the id is 1 then Buffer2 was used. In the case where data is not available WhichOne will equal -1. This routine will always return ApiSuccess (512) when either data has been transferred or when WhichOne = -1. A return value of ApiFailed (513) indicates that all the Records Per Buffer has been transferred.

Syntax

C/C++

```
RETURN_CODE
AlazarGetNextAutoDMABuffer(
      HANDLE h,
      void* Buffer1,
      void* Buffer2,
      long* WhichOne,
      long* RecordsTransferred,
      AUTODMA_STATUS* error,
      U32 r1,
      U32 r2,
      long *TriggersOccurred,
      U32 * r4
);
```

Parameters

h

[in] Handle to the device.

Buffer1

[out] This Buffer is used to transfer a complete set of Records from the Device back to the user application. It is one of two buffers that are alternated between. The second buffer is Buffer2.

Buffer1 should be large enough to contain (RecordsPerBuffer*TransferLength) many 16-bit values (VB-Integer, C&C++-short).

If the Record header is selected (UseHeader = 1) then Buffer1 should be large enough to hold (RecordsPerBuffer*(TransferLength+sizeof(ALAZAR_HEADER)) many 16bit values.

Buffer2

[out] This Buffer is used to transfer a complete set of Records from the Device back to the user. It is one of two buffers that are alternated between. The other buffer is Buffer1.

Buffer2 should be large enough to contain (RecordsPerBuffer*TransferLength) many 16-bit values (VB-Integer, C&C++-short).

If the Record header is selected (UseHeader = 1) then Buffer2 should be large enough to hold.

WhichOne

[out] This is a return value that indicates to the user which of the two Buffers (Buffer1 or Buffer2) the data was transferred into.

RecordsTransferred

[in | out] Indicates how many records have been transferred. This value will always be a multiple of RecordsPerBuffer. It is the application's responsibility to initialize the variable to 0 prior to the first call.

Error

[out] Error code.

This error code may be one of the following values.

Identifier	Value	Meaning
ADMA_Completed	0	No errors occurred
ADMA_Success	0	No errors occurred
ADMA_Buffer1Invalid	1	Buffer1 is not a suitable buffer
ADMA_Buffer2Invalid	2	Buffer2 is not a suitable buffer
ADMA_BoardHandleInvalid	3	Board handle is not valid
ADMA_InternalBuffer1Invalid	4	The routine cannot allocate enough memory because system resources are low
ADMA_InternalBuffer2Invalid	5	The routine cannot allocate enough memory because system resources are low
ADMA_OverFlow	6	A hardware overflow occurred
ADMA_InvalidChannel	7	The channel selected is invalid
ADMA_DMAInProgress	8	A memory transfer is in progress
ADMA_UseHeaderNotSet	9	UseHeader must be set
ADMA_HeaderNotValid	10	An invalid header was encountered
ADMA_InvalidRecsPerBuffer	11	RecordCount must be a perfect multiple of RecsPerBuffer

r1

[in] RESERVED.

r2

[in] RESERVED.

TriggersOccurred

[out] This is the total number of triggers that have been captured since the last start capture.

r4

[out] RESERVED.

Return value

See Table 1 for a list of error codes.

Remarks

Both Buffer1 and Buffer2 will be used in transferring the data from the device back to the user application. However, if the RecordsPerBuffer is set in conjunction with TransferLength such that all the data will fit in only one Buffer, then Only Buffer1 will be used and the WhichOne variable will equal 0. Only one transaction will take place. RecordsTransferred will be modified by the routine and is used to accumulate the number of record that has been transferred. Always set the variable to 0 before calling this routine and never modify its contents between repeating calls.

The user must ensure that Buffer1 and Buffer2 are valid buffers.

Buffer1 and Buffer2 should be large enough to contain (RecordsPerBuffer*TransferLength) many 16-bit values (VB-Integer, C&C++-short). If the Record header is selected (UseHeader = 1) then Buffer1 and Buffer2 should be large enough to hold (RecordsPerBuffer*(TransferLength+sizeof(ALAZAR_HEADER)) many 16bit values (VB-Integer, C&C++-short).

AlazarGetNextBuffer and **AlazarGetNextAutoDMABuffer** are identical.

See Also

AlazarStartAutoDMA
AlazarAbortAutoDma
AlazarGetNextBuffer
Using synchronous AutoDMA

4.1.55 AlazarGetNextBuffer

This function is deprecated. Do not use in new designs.

AlazarGetNextBuffer and AlazarGetNextAutoDMABuffer are identical. Please refer to AlazarGetNextAutoDMABuffer.

Syntax

```
C/C++
RETURN_CODE
AlazarGetNextBuffer (
      HANDLE h,
      void* Buffer1,
      void* Buffer2,
      long* WhichOne,
      long* RecordsTransferred,
      AUTODMA_STATUS* error,
      U32 r1,
      U32 r2,
      long *TriggersOccurred,
      U32 * r4
);
```

Remarks

AlazarGetNextBuffer and AlazarGetNextAutoDMABuffer are identical.

See Also

See AlazarGetNextAutoDMABuffer.

4.1.56 AlazarGetParameter

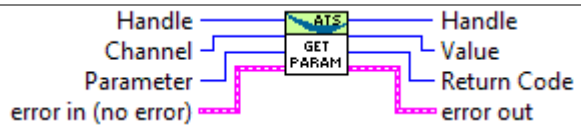
Get a device attribute as a signed long value.

Syntax

C/C++

```
RETURN_CODE
AlazarGetParameter(
      HANDLE BoardHandle,
      U8 Channel
      U32 Parameter,
      long *Value
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

Channel

[in] The channel of the attribute. This channel identifier may be a combination of one or more of the following values:

Identifier	Value	Description
CHANNEL_A	1	Apply parameter to channel A
CHANNEL_B	2	Apply parameter to channel B
CHANNEL_C	4	Apply parameter to channel C
CHANNEL_D	8	Apply parameter to channel D
CHANNEL_E	16	Apply parameter to channel E
CHANNEL_F	32	Apply parameter to channel F
CHANNEL_G	64	Apply parameter to channel G
CHANNEL_H	128	Apply parameter to channel H
CHANNEL_ALL	0	Apply parameter to all channels

Parameter

[in] The parameter identifier. Parameter identifiers include the following values:

Value	Meaning
DATA_WIDTH 0x10000009	Get the number of bits per sample.
SETGET_ASYNC_BUFFSIZE_BYTES 0x10000039	Get the size in bytes of each API allocated DMA.

SETGET_ASYNC_BUFFCOUNT 0x10000040	Get the number of API allocated DMA buffers.
GET_DATA_FORMAT 0x10000042	Return 0 if the data format is unsigned, or 1 if the data format is signed.
GET_SAMPLES_PER_TIMESTAMP_CLOCK 0x10000044	Get the number of sample clocks per timestamp clock
GET_RECORDS_CAPTURED 0x10000045	Get the current number of number of records captured since the start of the acquisition (single-port) or buffer (dual-port).
GET_ASYNC_BUFFERS_PENDING 0x10000050	Get the number of DMA buffers that have been queued by an application to this board.
GET_ASYNC_BUFFERS_PENDING_FULL 0x10000051	Get the number of DMA buffers for this board that are full and waiting to be processed by the application.
GET_ASYNC_BUFFERS_PENDING_EMPTY 0x10000052	Get the number of DMA buffers for this board that are empty and waiting to be filled by the board.
ECC_MODE 0x10000048	Get ECC mode: 0 = ECC is disabled, 1 = ECC enabled.
GET_AUX_INPUT_LEVEL 0x10000049	Read the TTL level of the AUX connector when it is configured as a serial input. 0 = TTL low, 1 = TTL high.
GET_CHANNELS_PER_BOARD 0x10000070	Get the number of analog input channels supported by the digitizer.
PACK_MODE 0x10000072	Get the data packing mode. 0 = PACK_DEFAULT, 1 = PACK_8_BITS_PER_SAMPLE, 2 = PACK_12_BITS_PER_SAMPLE
GET_FPGA_TEMPERATURE 0x10000080	Get the digitizer's current FPGA temperature in degrees Celcuis. This function is only supported on PCIe digitizers.

Value

[out] The parameter's value.

Return value

The function returns ApiSuccess (512) if it was able to retrieve the value of the specified parameter.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

See Also

AlazarGetParameterUL
AlazarSetParameter
AlazarSetParameterUL

4.1.57 AlazarGetParameterUL

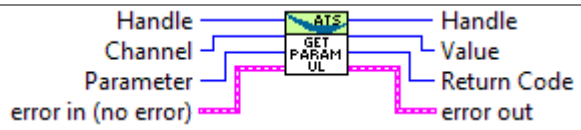
Get a device attribute as an unsigned 32-bit value.

Syntax

C/C++

```
RETURN_CODE
AlazarGetParameterUL (
      HANDLE BoardHandle,
      U8 Channel
      U32 Parameter,
      U32 *Value
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

Channel

[in] The channel of the attribute. This channel identifier may be a combination of one or more of the following values:

Identifier	Value	Description
CHANNEL_A	1	Apply parameter to channel A
CHANNEL_B	2	Apply parameter to channel B
CHANNEL_C	4	Apply parameter to channel C
CHANNEL_D	8	Apply parameter to channel D
CHANNEL_E	16	Apply parameter to channel E
CHANNEL_F	32	Apply parameter to channel F
CHANNEL_G	64	Apply parameter to channel G
CHANNEL_H	128	Apply parameter to channel H
CHANNEL_ALL	0	Apply parameter to all channels

Parameter

[in] The parameter identifier.

See AlazarApi.h for a complete list of list of identifiers.

Value

[out] The parameter's value.

Return value

The function returns ApiSuccess (512) if it was able to retrieve value of the specified parameter.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

See AlazarApi.h for a complete list of list of parameter identifiers.

See Also

AlazarGetParameter
AlazarSetParameter
AlazarSetParameterUL

4.1.58 AlazarGetSDKVersion

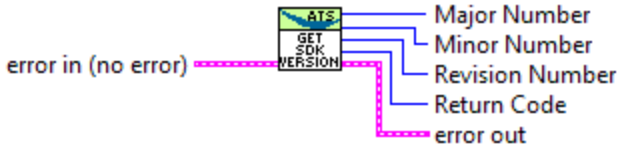
Get the driver library version. This is the version of ATSApi.dll under Windows, or ATSApi.so under Linux.

Syntax

C/C++

```
RETURN_CODE
AlazarGetSDKVersion (
      U8 *MajorNumber,
      U8 *MinorNumber,
      U8 *RevisionNumber
);
```

LabVIEW



Parameters

MajorNumber
[out] The SDK major version number.

MinorNumber
[out] The SDK minor version number.

RevisionNumber
[out] The SDK revision number.

Return value

The function returns ApiSuccess (512) if it was successful.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

Note that the version number returned is that of the driver library file, not the ATS-SDK version number.

SDK releases are given a version number with the format X.Y.Z where: X is the major release number, Y is the minor release number, and Z is the minor revision number.

See Also

AlazarGetCPLDVersion
AlazarGetDriverVersion

4.1.59 AlazarGetStatus

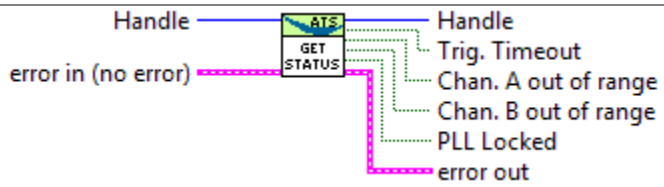
Return a bitmask with board status information.

Syntax

C/C++

```
U32 AlazarGetStatus (
      HANDLE BoardHandle
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

Return value

If the function fails, the return value is 0xffffffff.

If the function succeeds, the return value is contains board attributes. The attributes can include one or more of the following values.

Attribute	Meaning
1	At least one trigger timeout occurred.
2	At least one CHA sample was out of range during the last acquisition.
4	At least one CHB sample was out of range during the last acquisition
8	The PLL is locked (ATS660 only).

Remarks

See Also

4.1.60 AlazarGetSystemHandle

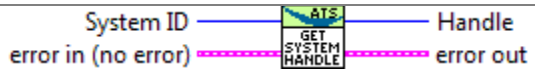
Return the handle of the master board in the specified board system.

Syntax

C/C++

```
HANDLE
AlazarGetSystemHandle (
      U32 SystemId
);
```

LabVIEW



Parameters

SystemId

[in] System identification number.

Return value

If this function succeeds, it returns a handle to the master board in the specified board system.

If the function fails, it returns NULL.

Remarks

If the board system specified contains a single, independent board, this function returns a handle to that board.

See Also

AlazarBoardsInSystemByHandle
AlazarBoardsInSystemBySystemID

4.1.61 AlazarGetTriggerAddress

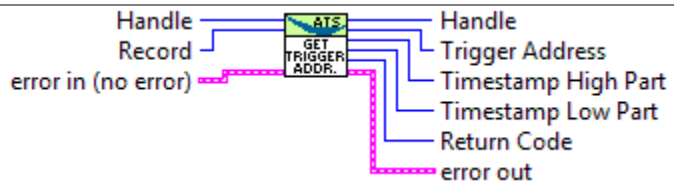
Get the timestamp and trigger address of the trigger event in a record acquired to on-board memory.

Syntax

C/C++

```
RETURN_CODE
AlazarGetTriggerAddress (
      HANDLE BoardHandle,
      U32 Record,
      U32 *TriggerAddress,
      U32 *TimestampHighPart,
      U32 *TimestampLowPart
);
```

LabVIEW



Parameters

BoardHandle
[in] Handle to board.

Record
[in] Record in acquisition (1-indexed).

TriggerAddress
[out] The trigger address.

TimestampHighPart
[out] The most significant 32-bits of a record timestamp.

TimestampLowPart
[out] The least significant 8-bits of a record timestamp.

Return value

The function returns ApiSuccess (512) if it was successful.

The function returns ApiError (604) if it is called after completing a dual-ported AutoDMA acquisition. It should be called after completing a single-ported acquisition.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

This function is part of the single-port data acquisition API. It cannot be used to retrieve the timestamp of records acquired using dual-port AutoDMA APIs.

The following code fragment demonstrates how to convert the trigger timestamp returned by **AlazarGetTriggerAddress** from counts to seconds.

```
__int64 timeStamp_cnt;
timeStamp_cnt = ((__int64) timestampHighPart) << 8;
timeStamp_cnt |= timestampLowPart & 0x0ff;

double samplesPerTimestampCount = 2; // board specific constant
double samplesPerSec = 50.e6;        // sample rate
double timeStamp_sec = (double) samplesPerTimestampCount *
    timeStamp_cnt / samplesPerSec;
```

Refer to *Table 2- Samples per Timestamp* for the board specific "sample clocks per timestamp count" values.

Example

See "%ATS_SDK_DIR%\Samples\SinglePort\AR_Timestamp" for a complete sample program demonstrates how to use AlazarGetTriggerAddress and convert the timestamp value to seconds.

See Also

AlazarRead
AlazarHyperDisp

4.1.62 AlazarGetTriggerTimestamp

Retrieve the timestamp, in sample clock periods, of a record acquired to on-board memory.

Syntax

```
C/C++
RETURN_CODE
AlazarGetTriggerTimestamp (
      HANDLE BoardHandle,
      U32 Record,
      U64 *Timestamp_samples
);
```

Parameters

BoardHandle
[in] Handle to board.

Record
[in] Record in acquisition (1-indexed).

Timestamp
[out] Record timestamp, in sample clock periods.

Return value

The function returns ApiSuccess (512) and if it was successful.

The function returns ApiError (604) if the record parameter is greater than 1000 in a dual-port AutoDMA acquisition.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

This function is part of the single-port data acquisition API. It cannot be used to retrieve the timestamp of records acquired using dual-port AutoDMA APIs.

Divide by the trigger timestamp value in sample clock periods by the sample rate to obtain the trigger timestamp value in seconds. For example:

```
// Get the trigger timestamp of the first record in sample clocks
U64 timestmap_samples;
AlazarGetTriggerTimestamp(handle, 1, &timestamp_samples);

// Convert the timestamp from sample clocks to seconds
double samplesPerSec = 100.e6;
double timestamp_seconds = timestmap_samples / samplesPerSec;
```

See Also

AlazarRead
AlazarHyperDisp

4.1.63 AlazarGetWhoTriggeredBySystemHandle

Return which event caused a board system to trigger and capture a record to on-board memory.

Syntax

C/C++

```
U32
AlazarGetWhoTriggeredBySystemHandle (
      HANDLE BoardHandle,
      U32 BoardId,
      U32 RecordNumber
);
```

LabVIEW

Parameters

BoardHandle

[in] Handle to master board in a board system.

BoardId

[in] Board identifier of a board in the specified board system.

RecordNumber

[in] Record in acquisition (1-indexed).

Return value

The function returns one of the following values.

Value	Meaning
0	This board did not cause the system to trigger.
1	CH A on this board caused the system to trigger.
2	CH B on this board caused the system to trigger.
3	EXT TRIG IN on this board caused the system to trigger.
4	Both CH A and CH B on this board caused the system to trigger.
5	Both CH A and EXT TRIG IN on this board caused the system to trigger.
6	Both CH B and EXT TRIG IN on this board caused the system to trigger.
7	A trigger timeout on this board caused the system to trigger.

Remarks

This function is part of the single-port data acquisition API. It cannot be used with the dual-port AutoDMA APIs.

© 2003-2015 Alazar Technologies Inc. Version 7.1.1

This API routine will not work with ATS850 version 1.2 hardware. Version 1.3 and higher version number of ATS850 are fully supported, as are all versions of ATS330 and ATS310.

See Also

AlazarGetWhoTriggeredBySystemId

4.1.64 AlazarGetWhoTriggeredBySystemID

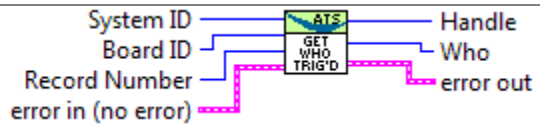
Return which event caused a board system to trigger and capture a record to on-board memory.

Syntax

C/C++

```
U32
AlazarGetWhoTriggeredBySystemID (
      U32 SystemId,
      U32 BoardId,
      U32 RecordNumber
);
```

LabVIEW



Parameters

SystemId

[in] System identifier number.

BoardId

[in] Board identifier of a board in the specified board system.

RecordNumber

[in] Record in acquisition (1-indexed).

Return value

The function returns one of the following values.

Value	Meaning
0	This board did not cause the system to trigger.
1	CH A on this board caused the system to trigger.
2	CH B on this board caused the system to trigger.
3	EXT TRIG IN on this board caused the system to trigger.
4	Both CH A and CH B on this board caused the system to trigger.
5	Both CH A and EXT TRIG IN on this board caused the system to trigger.
6	Both CH B and EXT TRIG IN on this board caused the system to trigger.
7	A trigger timeout on this board caused the system to trigger.

Remarks

This function is part of the single-port data acquisition API. It cannot be used with the dual-port AutoDMA APIs.

Note that this API routine will not work with ATS850 version 1.2 hardware. Version 1.3 and higher version number of ATS850 are fully supported, as are all versions of ATS330 and ATS310.

See Also

AlazarGetWhoTriggeredBySystemHandle

4.1.65 AlazarHyperDisp

Enable the on-board FPGA to process records acquired to on-board memory, and transfer the processed data to host memory.

Syntax

```
C/C++
RETURN_CODE
AlazarHyperDisp(
      HANDLE  BoardHandle,
      void   *Buffer,
      U32     BufferSize,
      U8     *ViewBuffer,
      U32     ViewBufferSize,
      U32     NumOfPixels,
      U32     Option,
      U32     ChannelSelect,
      U32     Record,
      long    TransferOffset,
      U32     *Error
);
```

Parameters

BoardHandle

[in] Handle to a board.

Buffer

[in] Reserved (set to NULL).

BufferSize

[in] Number of samples to process.

ViewBuffer

[out] Pointer to a buffer to receive processed data.

ViewBufferSize

[in] Size, in bytes, of processed data buffer.

NumOfPixels

[in] Number of HyperDisp points.

Option

[in] Processing mode.

Value	Meaning
1	Enable HyperDisp processing.

ChannelSelect

[in] Channel to process.

Record

[in] Record to process (1-indexed).

TransferOffset
[in] Offset, in samples, of first sample to process relative to trigger position in record.

Error
[out] Pointer to value to receive a result code.

Return values

If the function succeeds, it returns ApiSuccess (512).

If the function fails, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

AlazarHyperDisp is part of the single-port data acquisition API. It cannot be used with the dual-port AutoDMA APIs.

HyperDisp processing enables the on-board FPGA to divide a record acquired to on-board memory into intervals, find the minimum and maximum sample values during each interval, and transfer an array of minimum and maximum sample values to a buffer in host memory. This allows the acquisition of relatively long records to on-board memory, but the transfer of relatively short, processed records to a buffer in host memory.

For example, it would take an ATS860-256M about ~2.5 seconds to transfer a 250,000,000 sample record from on-board memory, across the PCI bus, to a buffer in host memory. With HyperDisp enabled, it would take the on-board FPGA a fraction of a second to process the record and transfer a few hundred samples from on-board memory, across the PCI bus, to a buffer in host memory.

Example

The "%ATS_SDK_DIR%\SinglePort\HD" sample program demonstrates how to use the **AlazarHyperDisp** API.

See Also

AlazarGetTriggerAddress
AlazarRead
AlazarReadEx

4.1.66 AlazarInputControl

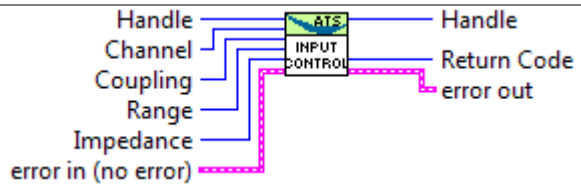
Select the input coupling, range, and impedance of a digitizer channel.

Syntax

C/C++

```
RETURN_CODE
AlazarInputControl (
      HANDLE BoardHandle,
      U8 ChannelId,
      U32 CouplingId,
      U32 RangeId,
      U32 ImpedanceId
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to a board.

ChannelId

[in] Select the channel to control. This parameter may be one of the following identifiers or values.

Identifier	Value
CHANNEL_A	1
CHANNEL_B	2
CHANNEL_C	4
CHANNEL_D	8
CHANNEL_E	16
CHANNEL_F	32
CHANNEL_G	64
CHANNEL_H	128
CHANNEL_I	256
CHANNEL_J	512
CHANNEL_K	1024
CHANNEL_L	2048
CHANNEL_M	4096
CHANNEL_N	8192
CHANNEL_O	16384
CHANNEL_P	32768

© 2003-2015 Alazar Technologies Inc. Version 7.1.1

CouplingId

[in] Specify coupling of selected channel. This parameter may be one of the following identifiers or values. Please refer to the board's datasheets for information on available input couplings.

Identifier	Value
AC_COUPLING	1
DC_COUPLING	2

RangeId

[in] Specify full-scale input range of selected channel. See *Table 5 - Available Impedances and Input Ranges* for a list of available input ranges.

ImpedanceId

[in] Specify termination of selected channel. This parameter may be one of the following identifiers or values. See *Table 5 - Available Impedances and Input Ranges* for more details.

Identifier	Value
IMPEDANCE_1M_OHM	1
IMPEDANCE_50_OHM	2

Return values

If the function succeeds, it returns ApiSuccess (512).

If the digitizer board does not support the specified input range, coupling, or the impedance, the function returns ApiFailed (513).

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 Error Codes for a list of error codes.

See Also

AlazarSetBWLimit

4.1.67 AlazarInputControlEx

Select the input coupling, range, and impedance of a digitizer channel. This function takes the same parameters as AlazarInputControl, except that values larger than one byte for the ChannelId are accepted. Please see AlazarInputControl's documentation for more information.

Syntax

C/C++

```
RETURN_CODE
AlazarInputControl (
      HANDLE BoardHandle,
      U32 ChannelId,
      U32 CouplingId,
      U32 RangeId,
      U32 ImpedanceId
);
```

4.1.68 AlazarNumOfSystems

Get the total number of board systems detected.

Syntax

```
C/C++
U32 AlazarNumOfSystems ();
```

Parameters

None

Return value

The total number of board systems detected.

Remarks

A *board system* is a group of one or more digitizer boards that share clock and trigger signals. A board system may be composed of a single independent board, or a group of two or more digitizer boards connected together with a SyncBoard.

See Also

AlazarBoardsInSystemByHandle
AlazarBoardsInSystemBySystemID

4.1.69 AlazarOEMDownLoadFPGA

Download an FPGA image to a digitizer board.

Syntax

```
C/C++
RETURN_CODE
AlazarOEMDownLoadFPGA(
      HANDLE BoardHandle,
      char *FileName,
      U32 *Error
);
```

Parameters

BoardHandle
[in] Handle to a board.

FileName
[in] FPGA image file path.

Error
[out] Download result.

Return value

If the function succeeds, it returns ApiSuccess (512).

If the function fails, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

See Also

AlazarParseFPGAName

4.1.70 AlazarOpen

This function is obsolete. Do not use in new designs.

Open and initialize a board.

Syntax

```
C/C++
HANDLE
AlazarOpen(
      char *BoardName
);
```

Parameters

BoardName

[in] Name of board created by driver. For example "ATS850-0".

Return value

A handle to the board.

Remarks

The ATS library manages board handles internally. This function should only be used in applications that are written for single board digitizer systems.

See Also

AlazarClose

4.1.71 AlazarParseFPGAName

Decode an OEM FPGA image file name.

Syntax

```
C/C++
RETURN_CODE
AlazarParseFPGAName (
      const char *FilePath,
      char *FileName,
      U32 *BoardType,
      U32 *MemorySizeId,
      U32 *HardwareMajorVersion,
      U32 *HardwareMinorVersion,
      U32 *FpgaMajorVersion,
      U32 *FpgaMinorVersion,
      U32 *Error
);
```

Parameters

FilePath
[in] Full path to FPGA image file.

FileName
[out] FPGA image file name.

MemorySizeId
[out] The memory size identifier of the memory in samples per channel required on the digitizer board.

HardwareMajorVersion
[out] Pointer to digitizer board major version number.

HardwareMinorVersion
[out] Pointer to digitizer board minor version number.

FpgaMajorVersion
[out] Pointer to FPGA major version number.

FpgaMinorVersion
[out] Pointer to FPGA minor version number.

Error
[out] Pointer to an error code.

Return value

If the function succeeds, it returns ApiSuccess (512).

If the path to the file path was not found, the function returns ApiFailed (513) and Error to 626.

Remarks

See Also

AlazarOEMDownLoadFPGA

4.1.72 AlazarPostAsyncBuffer

Add a buffer to the end of a list of buffers available to be filled by the board. Use AlazarWaitAsyncBufferComplete to determine if the board has received sufficient trigger events to fill this buffer.

Syntax

```
C/C++
RETURN_CODE
AlazarPostAsyncBuffer (
   HANDLE BoardHandle,
   void *Buffer,
   U32 BufferLength
);
```

Parameters

BoardHandle
[in] Handle to board.

Buffer
[out] Pointer to a buffer to receive sample data from the digitizer board.

BufferLength
[in] Specifies the length of the buffer in bytes.

Return values

If the function succeeds in adding the buffer to end of the list of buffers available to be filled by the board, it returns ApiSuccess (512). Use AlazarWaitAsyncBufferComplete to determine when the board has received sufficient trigger events to file this buffer.

If the function fails, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

You must call AlazarBeforeAsyncRead before calling **AlazarPostAsyncBuffer**.

You must call AlazarAbortAsyncRead before your application exits if you have called **AlazarPostAsyncBuffer**, and buffers are pending when your application exits.

The BufferLength parameter must be equal to the product of the number of bytes per record, the number of records per buffer, and the number of enabled channels. If record headers are enabled, the number of bytes per record must include the size of the record header (16 bytes).

See Also

AlazarAbortAsyncRead

AlazarBeforeAsyncRead
Using asynchronous AutoDMA

4.1.73 AlazarQueryCapability

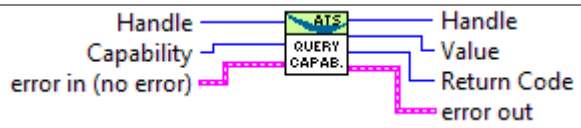
Get a device attribute.

Syntax

C/C++

```
RETURN_CODE
AlazarQueryCapability (
      HANDLE BoardHandle,
      U32 Capability,
      U32 Reserved,
      U32 *Value
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

Capability

[in] Capability identifier. See remarks below.

Reserved

[in] Reserved (Set to 0).

Value

[out] Capability value.

Return value

The function returns ApiSuccess (512) if it was able to retrieve value of the specified capability.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

Capability identifiers include the following items. See AlazarApi.h for a complete list of list of capability identifiers.

Identifier	Value	Description
GET_SERIAL_NUMBER	0x10000024	Return the board serial number
GET_LATEST_CAL_DATE	0x10000026	Return the board's latest calibration date as a decimal

		number with the format DDMMYY where DD is 1-31, MM is 1-12, and YY is 00-99 from 2000.
GET_LATEST_CAL_DATE_MONTH	0x1000002D	Return the month of the board's latest calibration date as a decimal number with the format MM where M is 1-12.
GET_LATEST_CAL_DATE_DAY	0x1000002E	Return the day of month of the board's latest calibration date as a decimal number with the format DD where DD is 1-31.
GET_LATEST_CAL_DATE_YEAR	0x1000002F	Return the year of the board's latest calibration date as a decimal number with the format YY where YY is 00-99 from 2000.
MEMORY_SIZE	0x1000002A	Return the on-board memory size in maximum samples per channel in single channel mode. See AlazarGetChannelInfo for more information.
BOARD_TYPE	0x1000002B	Return the board type identifier. See AlazarGetBoardKind for more information.
ASOPC_TYPE	0x1000002C	Return the board's FPGA signature.
GET_PCIE_LINK_SPEED	0x10000030	Return the PCIe link speed negotiated between a PCIe digitizer board and the host PCIe bus in 2.5G bits per second units. The PCIe bus uses 10b/8b encoding, so divide the link speed by 10 to find the link speed in bytes per second. For example, a link speed of 2.5 Gb/s gives 250 MB/s per lane. PCIe Gen 2 digitizers such as ATS9360 should receive 5.0 Gb/s links.

		PCIe Gen 3 digitizers such as ATS9373 should receive 8.0 Gb/s links.
GET_PCIE_LINK_WIDTH	0x10000031	Return the PCIe link width in lanes negociated between a PCIe digitizer board and the host PCIe bus. An ATS9462 should negociate 4 lanes, while the ATS9325, ATS9350, ATS9351, ATS9360, ATS9373, ATS9440, ATS9850 and ATS9870 should negociate 8 lanes. If a board obtains fewer lanes, then the board may be installed in a PCIe slot that does not support the expected number of lanes. The ideal PCIe bandwidth is the link speed in bytes per second per lane, multiplied by the link width in lanes. For example, and ATS9870 that negociates 8 lanes at 250 MB/s per lane has an ideal bandwidth of 2 GB/s.
GET_MAX_PRETRIGGER_SAMPLES	0x10000046	Get the maximum number of pre-trigger samples.
GET_CPF_DEVICE	0x10000071	Get the model of user-programable FPGA device. 1 = CPF_DEVICE_EP3SL50, 2 = CPF_DEVICE_EP3SE260

See Also

AlazarGetBoardKind
AlazarGetChannelInfo
AlazarGetParameter
AlazarGetParameterUL

4.1.74 AlazarRead

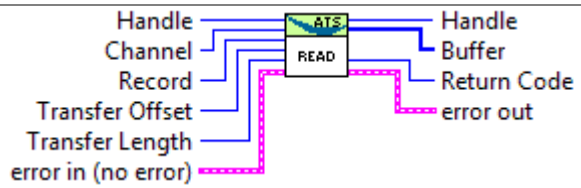
Read all or part of a record from an acquisition to on-board memory from on-board memory to a buffer in host memory. The record must be less than 2,147,483,648 samples long.

Syntax

C/C++

```
U32
AlazarRead (
      HANDLE BoardHandle,
      U32 ChannelId,
      void *Buffer,
      int ElementSize,
      long Record,
      long TransferOffset,
      U32 TransferLength
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to a board.

ChannelId

[out] Channel identifier of record.

Buffer

[out] Buffer to receive sample data.

ElementSize

[in] Number of bytes per sample.

Record

[in] Record in on-board memory to transfer to buffer (1-indexed).

TransferOffset

[in] The offset, in samples from the trigger position in the record, of the first sample in the record in on-board memory to transfer into the buffer.

TransferLength

[in] The number of samples to transfer from the record in on-board memory into the buffer.

Return values

If the function succeeds, it returns ApiSuccess (512).

If the function fails, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

AlazarRead is part of the single-port data acquisition API. It cannot be used with the dual-port AutoDMA APIs.

AlazarRead can transfer segments of a record acquired to on-board memory. This may be useful if a full record is too large to transfer as a single block, or if only part of a record is of interest.

Use either **AlazarRead** or AlazarReadEx to transfer records with less than 2,147,483,648 samples. Use AlazarReadEx to transfer records with 2,147,483,648 or more samples.

Examples

The "%ATS_SDK_DIR%\Samples\SinglePort\AR" sample program demonstrates how to use **AlazarRead**.

The "%ATS_SDK_DIR%\Samples\SinglePort\AR_Segments" sample program demonstrates how to use **AlazarRead** to split records in to segments for transfer from on-board to host memory.

See Also

AlazarHyperDisp
AlazarGetTriggerAddress
AlazarReadEx
Using AlazarRead

4.1.75 AlazarReadEx

Read all or part of a record from an acquisition to on-board memory from on-board memory to a buffer in host memory. The record may be 2,147,483,648 or more samples long.

Syntax

```
C/C++
U32
AlazarReadEx (
      HANDLE BoardHandle,
      U32 ChannelId,
      void *Buffer,
      int ElementSize,
      long Record,
      INT64 TransferOffset,
      U32 TransferLength
);
```

Parameters

BoardHandle
[in] Handle to a board.

ChannelId
[out] Channel identifier of record.

Buffer
[out] Buffer to receive sample data.

ElementSize
[in] Number of bytes per sample.

Record
[in] Record in on-board memory to transfer to buffer (1-indexed).

TransferOffset
[in] The offset, in samples from the trigger position in the record, of the first sample in the record in on-board memory to transfer into the buffer.

TransferLength
[in] The number of samples to transfer from the record in on-board memory into the buffer.

Return values

If the function succeeds, it returns ApiSuccess (512).

If the function fails, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

AlazarReadEx is part of the single-port data acquisition API. It cannot be used with the dual-port AutoDMA APIs.

AlazarReadEx can transfer segments of a record acquired to on-board memory. This may be useful if a full record is too large to transfer as a single block, or if only part of a record is of interest.

Use AlazarRead or **AlazarReadEx** to transfer records with less than 2,147,483,648 samples. Use **AlazarReadEx** to transfer records with 2,147,483,648 or more samples.

Examples

See Also

AlazarRead
AlazarHyperDisp
AlazarGetTriggerAddress
Using AlazarRead

4.1.76 AlazarResetTimeStamp

Control record timestamp counter resets.

Syntax

C/C++

```
RETURN_CODE
AlazarResetTimeStamp(
      HANDLE BoardHandle,
      U32 Option
);
```

LabVIEW

Parameters

BoardHandle

[in] Handle to board.

Option

[in] Record timestamp counter reset options. The option can be one of the following values. See AlazarApi.h for a complete list.

Identifier	Description
TIMESTAMP_RESET_FIRSTTIME_ONLY (0)	Reset the timestamp counter to zero on the next call to AlazarStartCapture, but not thereafter.
TIMESTAMP_RESET_ALWAYS (1)	Reset the timestamp counter to zero on each call to AlazarStartCapture. This is the default operation.

Return value

If the function succeeds, it returns ApiSuccess (512).

If the function fails, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

This function is not supported by the ATS310, ATS330, and ATS850.

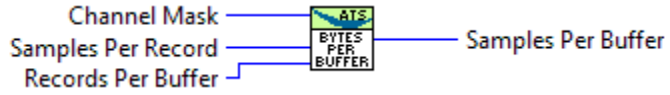
See Also

4.1.77 AlazarSamplesPerBuffer

Computes the bytes per buffer given the appropriate input. This function is a LabVIEW helper function only available in this language.

Syntax

LabVIEW



Parameters

Channel Mask

[in] The channel mask.

Samples Per Record

[in] The sum of pre- and post-trigger samples.

Records Per Buffer

[in] The number of records in each buffer.

Return value

The number of bytes in each buffer.

4.1.78 AlazarSetBWLimit

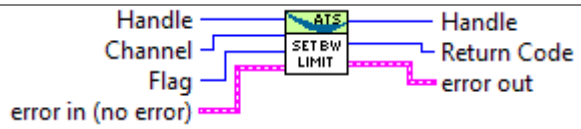
Control bandwidth of an input channel. This function is not available on all boards; see *Table 9 – Miscellaneous Features* Support.

Syntax

C/C++

```
RETURN_CODE
AlazarSetBWLimit(
      HANDLE BoardHandle,
      U32 ChannelId,
      U32 Flag
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

ChannelId

[in] Channel identifier.

Flag

[in] Enable bandwidth limit flag. The flag can be one of the following values.

Value	Description
0	Disable bandwidth limit.
1	Enable bandwidth limit.

Return value

If the function succeeds, it returns ApiSuccess (512).

If the function fails, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

The bandwidth limiter is disabled by default. When enabled, the bandwidth limiter reduces input bandwidth to approximately 20 MHz.

See Also

AlazarInputControl

4.1.79 AlazarSetCaptureClock

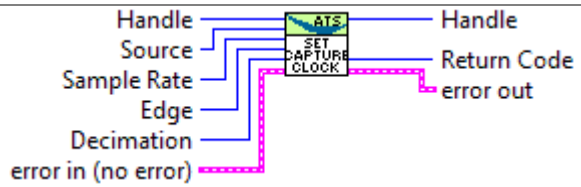
Configure sample clock source, edge, and decimation.

Syntax

C/C++

```
RETURN_CODE
AlazarSetCaptureClock(
      HANDLE BoardHandle,
      U32 SourceId,
      U32 SampleRateIdOrValue,
      U32 EdgeId,
      U32 Decimation
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

SourceId

[in] Clock source identifier. Valid values depend on the board model. See *Table 11 – Supported clock types* for a list of supported values. For external clock types, the identifier to choose may depend on the external clock's frequency. See *Table 12 - Frequency limits for external clock types (values in MHz unless noted otherwise)* for a list of frequency ranges for all clock types.

SampleRateIdOrValue

[in] If the clock source chosen is INTERNAL_CLOCK, this value is an identifier for the internal sample rate to choose. See *Table 6 - Sample Rates* for a list of available sample rates for each board. The sample rate must be specified via one of the identifiers supplied by the AlazarTech headers. See *Table 7 - Sample Rate Identifiers* for the identifiers corresponding to various sample rates.

If the clock source chosen is EXTERNAL_CLOCK_10_MHZ_REF, pass the value of the sample clock to generate from the 10 MHz reference in Hertz to *SampleRateIdOrValue*. The exact values that can be generated depend on the board model. See *Table 13 - Valid frequencies in PLL mode* for a list of valid frequencies.

Otherwise, the clock source is an external clock. Pass SAMPLE_RATE_USER_DEF to this parameter.

EdgeId

[in] Select the external clock edge on which to latch samples data. The clock edge identifier may be one of the following values.

Identifier	Value	Description
CLOCK_EDGE_RISING	0	Sample on rising edge of external clock.
CLOCK_EDGE_FALLING	1	Sample on falling edge of external clock.

Decimation

[in] Clock decimation value. See the remarks below.

Return value

If the function succeeds, it returns ApiSuccess (512).

If the function fails, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

The clock decimation value may be any integer between 0 and 100000 with the following exceptions. Note that a decimation value of 0 means disable decimation.

- If an ATS460/ATS660/ATS860 is configured to use a SLOW_EXTERNAL_CLOCK clock source, the maximum decimation value is 1.

- If an ATS9350 is configured to use an EXTERNAL_CLOCK_10MHz_REF clock source, the decimation value must be 1, 2, 4 or any multiple of 5. Note that the sample rate identifier value must be 500000000, and the sample rate will be 500 MHz divided by the decimation value.

- If an ATS9360 / ATS9373 is configured to use an EXTERNAL_CLOCK_10MHz_REF clock source, the maximum decimation value is 1.

- If an ATS9850 is configured to use an EXTERNAL_CLOCK_10MHz_REF clock source, the decimation value must be 1, 2, 4 or any multiple of 10. Note that the sample rate identifier value must be 500000000, and the sample rate will be 500 MHz divided by the decimation value.

- If an ATS9870 is configured to use an EXTERNAL_CLOCK_10MHz_REF clock source, the decimation value must be 1, 2, 4 or any multiple of 10. Note that the sample rate identifier value must be 1000000000, and the sample rate will be 1 GHz divided by the decimation value.

4.1.80 AlazarSetClockSwitchOver

Configure the dummy clock.

When the "dummy clock" option enabled, the digitizer uses an internally generated clock signal to drive its ADCs for a specified amount of time after the end of each record. At the end of the dummy clock on time, the digitizer switches back to using the external clock signal to clock the ADCs.

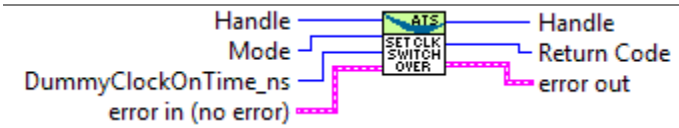
Scanning applications that generate an unusable external clock signal during horizontal retrace periods at the end of each scan line should consider enabling the "dummy clock" option.

Syntax

C/C++

```
RETURN_CODE
AlazarSetClockSwitchOver(
      HANDLE hBoard,
      U32    Mode,
      U32    DummyClockOnTime_ns,
      U32    Reserved
      )
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

Mode

[in] Clock switch over mode. The parameter can be one of the following values:

Name	Value	Description
CSO_DISABLE	0	Disable clock switch over.
CSO_ENABLE_DUMMY_CLOCK	1	Enable dummy clock mode.

DummyClockOnTime_ns

[in] Dummy clock on-time in nanoseconds.

If dummy clock mode is enabled, the digitizer switches to an internally generated dummy clock after capturing each record, and uses the dummy clock for this amount of time. After the dummy clock on-time period has expired, the digitizer switches back to using an external clock signal.

Reserved
[in] Not used.

Return value

If the function succeeds, it returns ApiSuccess (512).

If the feature is not implemented in firmware, the function returns ApiUnsupportedFunction. Please contact AlazarTech for updated firmware.

If the function fails, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

The dummy clock on-time should be selected so that it is longer than the amount time that external clock signal is unstable, but must not be longer than the time between the end of one record, and the start of the next record.

The dummy clock feature is implemented in the ATS9351, ATS9440, ATS9625, ATS9626, and ATS9350 with firmware version 14.10 or later.

See Also

AlazarSetCaptureClock

4.1.81 AlazarSetExternalClockLevel

Set the external clock comparator level.

Syntax

C/C++

```
RETURN_CODE
AlazarSetExternalClockLevel(
      HANDLE BoardHandle,
      float Level_percent
);
```

LabVIEW

Parameters

BoardHandle

[in] Handle to board.

Level_percent

[in] The external clock comparator level, in percent.

Return value

If the function succeeds, it returns ApiSuccess (512).

If the function fails, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Note that the function fails with error code ApiUnsupportedFunction (533) if the digitizer does not support setting the external clock comparator level. The following table lists the boards that support this feature.

Model	Supported
ATS310, ATS330, ATS850, ATS9360, ATS9373, ATS9416	No
ATS460, ATS660, ATS860, ATS9350, ATS9351, ATS9440, ATS9462, ATS9625, ATS9626, ATS9870, AXI9870	Yes

Remarks

See Also

AlazarSetCaptureClock

4.1.82 AlazarSetExternalTrigger

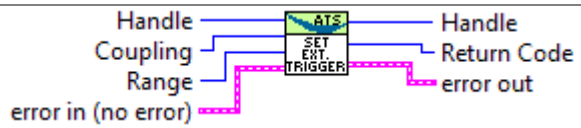
Set the external trigger range and coupling.

Syntax

C/C++

```
RETURN_CODE
AlazarSetExternalTrigger (
      HANDLE BoardHandle,
      U32 CouplingId,
      U32 RangeId
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

CouplingId

[in] Specifies the external trigger coupling. This parameter may have one of the following values.

Identifier	Value	Description
AC_COUPLING	1	AC coupled trigger input
DC_COUPLING	2	DC coupled trigger input

RangeId

[in] Specifies the external trigger range. This parameter may have one of the following values. See *Table 10- External Trigger Level Support* to see options supported by each board model.

Identifier	Value	Description
ETR_5V	0	±5V external trigger range.
ETR_1V	1	±1V external trigger range.
ETR_TTL	2	TTL level external trigger range
ETR_2V5	3	±2.5V external trigger range

Return value

If the function succeeds, it returns ApiSuccess (512).

If the function fails, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

See Also

AlazarSetTriggerDelay
AlazarSetTriggerOperation
AlazarSetTriggerTimeOut

4.1.83 AlazarSetLED

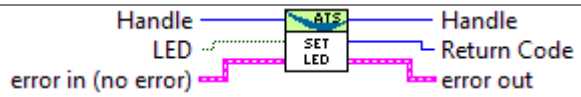
Control LED on a board's PCI/PCIe mounting bracket.

Syntax

C/C++

```
RETURN_CODE
AlazarSetLED (
      HANDLE BoardHandle,
      U32 LedOn
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

LedOn

[in] Specify LED state. This parameter may have one of the following values.

Identifier	Value	Description
LED_OFF	0	Turn off LED
LED_ON	1	Turn on LED

Return value

If the function succeeds, it returns ApiSuccess (512).

If the function fails, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

See the "%ATS_SDK_DIR%\Samples\AlazarSysInfo" for a sample program that controls the LED.

See Also

4.1.84 AlazarSetParameter

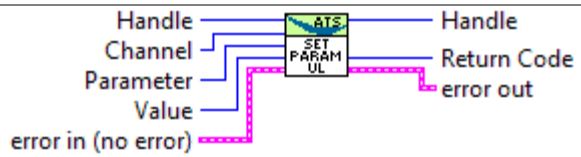
Set a device parameter as a signed long value.

Syntax

C/C++

```
RETURN_CODE
AlazarSetParameter(
      HANDLE BoardHandle,
      U8 ChannelId
      U32 ParameterId,
      long Value
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

ChannelId

[in] The channel to control. This channel identifier may be a combination of one or more of the following values:

Identifier	Value	Description
CHANNEL_A	1	Apply parameter to channel A
CHANNEL_B	2	Apply parameter to channel B
CHANNEL_C	4	Apply parameter to channel C
CHANNEL_D	8	Apply parameter to channel D
CHANNEL_E	16	Apply parameter to channel E
CHANNEL_F	32	Apply parameter to channel F
CHANNEL_G	64	Apply parameter to channel G
CHANNEL_H	128	Apply parameter to channel H
CHANNEL_ALL	0	Apply parameter to all channels

ParameterId

[in] Parameter to modify.

The parameter identifier may be one of the following values. See AlazarApi.h for a complete list of list of parameter identifiers.

Identifier	Value	Description
SETGET_ASYNC_BUFFCOUNT	0x10000040	Select number of API allocated DMA buffers.

SET_DATA_FORMAT	0x10000041	Select sample data format: 0 = unsigned , 1 = signed.
ECC_MODE	0x10000048	Set EEC mode: 0 = disable, 1 = enable.
PACK_MODE	0x10000072	Set the data packing mode. 0 = PACK_DEFAULT, 1 = PACK_8_BITS_PER_SAMPLE, 2 = PACK_12_BITS_PER_SAMPLE

Value

[in] Parameter value.

Return value

The function returns ApiSuccess (512) if it was able to retrieve the value of the specified parameter.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

See Also

AlazarGetParameter
AlazarGetParameterUL
AlazarSetParameterUL

4.1.85 AlazarSetParameterUL

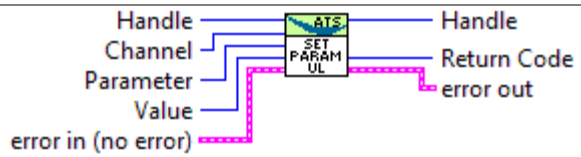
Set a device parameter as an unsigned 32-bit value.

Syntax

C/C++

```
RETURN_CODE
AlazarSetParameterUL (
      HANDLE BoardHandle,
      U8 ChannelId
      U32 ParameterId,
      U32 Value
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

ChannelId

[in] The channel to control. This channel identifier may be a combination of one or more of the following values:

Identifier	Value	Description
CHANNEL_A	1	Apply parameter to channel A
CHANNEL_B	2	Apply parameter to channel B
CHANNEL_C	4	Apply parameter to channel C
CHANNEL_D	8	Apply parameter to channel D
CHANNEL_E	16	Apply parameter to channel E
CHANNEL_F	32	Apply parameter to channel F
CHANNEL_G	64	Apply parameter to channel G
CHANNEL_H	128	Apply parameter to channel H
CHANNEL_ALL	0	Apply parameter to all channels

ParameterId

[in] Parameter to modify. The parameter identifier may be one of the following values. See AlazarApi.h for a complete list of list of parameter identifiers.

Identifier	Value	Description
SET_ADC_MODE	0x10000047	Set the ADC mode: 0 = default, 1 = dual-edge sampling

Value
[in] Parameter value.

Return value

The function returns ApiSuccess (512) if it was able to retrieve the value of the specified parameter.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

See AlazarApi.h for a list of parameter identifiers.

See Also

AlazarGetParameter
AlazarGetParameterUL
AlazarSetParameter

4.1.86 AlazarSetRecordCount

Select the number of records to capture to on-board memory.

Syntax

C/C++

```
RETURN_CODE
AlazarSetRecordCount (
      HANDLE BoardHandle,
      U32 RecordsPerCapture
);
```

LabVIEW

Parameters

BoardHandle

[in] Handle to board.

RecordsPerCapture

[in] The number of records to acquire to on-board memory during the acquisition.

Return value

The function returns ApiSuccess (512) if it was successful.

The function returns 607 if the number of records is greater than greater than the maximum number supported by the firmware revision.

Otherwise, the function returns an error code indicating the reason that it failed. See Table 1 for a list of error codes.

Remarks

This function is part of the single-port API. It cannot be used with the dual-port AutoDMA functions.

The maximum number of records per capture is a function of the board type, the maximum number of samples per channel (SPC_{max}), and the current number of samples per record (SPR):

Board type	Maximum records per capture
ATS850, ATS310, ATS330	SPC_{max} / (SPR + 16) or 10000, whichever is smaller.
ATS460, ATS660, ATS9462	SPC_{max} / (SPR + 16) or 256000 records, whichever is smaller.

ATS860, ATS9325, ATS9350, ATS9351	SPC_{max} / (SPR + 32) or 256000 records, whichever is smaller.
ATS9850, ATS9870	SPC_{max} / (SPR + 64) or 256000 records, whichever is smaller.

See Also

AlazarRead
AlazarHyperDisp
AlazarGetTriggerAddress

4.1.87 AlazarSetRecordSize

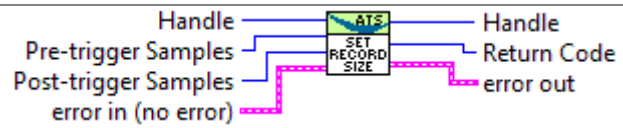
Set the number of pre- and post-trigger samples per record.

Syntax

C/C++

```
RETURN_CODE
AlazarSetRecordSize(
      HANDLE BoardHandle,
      U32 PreTriggerSamples,
      U32 PostTriggerSamples
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

PreTriggerSamples

[in] The number of samples before the trigger position in each record.

PostTriggerSamples

[in] The number of samples at or after the trigger position in each record.

Return value

The function returns ApiSuccess (512) if it was able to retrieve the value of the specified parameter.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

The number of pre-trigger samples must not exceed the number of samples per record minus 64.

The number of samples per record is the sum of the pre- and post-trigger samples. It must follow requirements specific to each board listed in *Table 8 - Samples per Record Alignment Requirements*.

See Also

AlazarBeforeAsyncRead
AlazarRead

AlazarStartAutoDMA

4.1.88 AlazarSetTriggerDelay

Set the time, in sample clocks, to wait after receiving a trigger event before capturing a record for the trigger.

Syntax

C/C++

```
RETURN_CODE
AlazarSetTriggerDelay (
      HANDLE BoardHandle,
      U32 Value
);
```

LabVIEW

Parameters

BoardHandle

[in] Handle to board.

Value

[in] Trigger delay in sample clocks.

Return value

The function returns ApiSuccess (512) if it was able to retrieve the value of the specified parameter.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

To convert the trigger delay from seconds to sample clocks, multiple the sample rate in samples per second by the trigger delay in seconds. For example, if the sample rate is 50 MS/s and the desired trigger delay is 1 ms, then the trigger delay in sample clocks is 50e6 samples per second x 1e-3 seconds = 50000 samples.

The trigger delay value may be 0 to 9,999,999 samples.

The trigger delay value must be a multiple of a certain value for each board. This is listed in *Table 3 - Trigger Delay Alignment.*

See Also

AlazarSetExternalTrigger
AlazarSetTriggerOperation

AlazarSetTriggerTimeOut

4.1.89 AlazarSetTriggerOperation

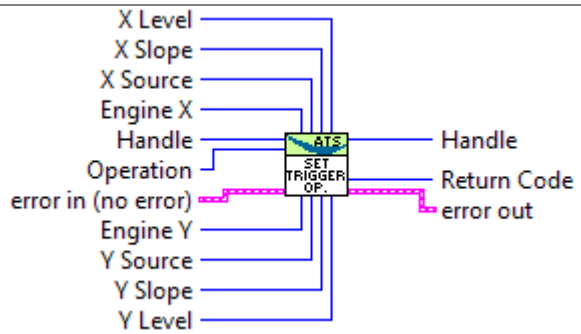
Configure the trigger system.

Syntax

C/C++

```
RETURN_CODE
AlazarSetTriggerOperation (
      HANDLE BoardHandle,
      U32 TriggerOperation,
      U32 TriggerEngineId1,
      U32 SourceId1,
      U32 SlopeId1,
      U32 Level1,
      U32 TriggerEngineId2,
      U32 SourceId2,
      U32 SlopeId2,
      U32 Level2
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

TriggerOperation

[in] Specify how the two independent trigger engines generate a trigger event.

This parameter can be one of the following values.

Identifier	Value	Meaning Generate a trigger event when…
TRIG_ENGINE_OP_J	0	T_j goes low to high.
TRIG_ENGINE_OP_K	1	T_k goes low to high.
TRIG_ENGINE_OP_J_OR_K	2	T_j goes low to high, or T_k goes low to high.
TRIG_ENGINE_OP_J_AND_K	3	(T_j AND $T_{k)}$ goes low to high.
TRIG_ENGINE_OP_J_XOR_K	4	(T_j XOR $T_{k)}$ goes low to high.
TRIG_ENGINE_OP_J_AND_NOT_K	5	(T_j AND (NOT T_k)) goes low to

		high.
TRIG_ENGINE_OP_NOT_J_AND_K	6	((NOT T_j)AND T_k) goes low to high.

Note that the symbol T_j represents the output of trigger engine J, and T_k represents the output of trigger engine K.

TriggerEngineId1
TriggerEngineId2

[in] Select the trigger engine to configure

This parameter can be one of the following values.

Identifier	Value	Description
TRIG_ENGINE_J	0	Configure trigger engine J
TRIG_ENGINE_K	1	Configure trigger engine K

SourceId1
SourceId2

[in] Select the signal source for the specified trigger engine.

This parameter can be one of the following values.

Identifier	Value	Description
TRIG_CHAN_A	0x0	Use signals from CH A
TRIG_CHAN_B	0x1	Use signals from CH B
TRIG_EXTERNAL	0x2	Use signals from the TRIG IN input
TRIG_DISABLE	0x3	Disable this trigger engine.
TRIG_CHAN_C	0x4	Use signals from CH C
TRIG_CHAN_D	0x5	Use signals from CH D
TRIG_CHAN_E	0x6	Use signals from CH E
TRIG_CHAN_F	0x7	Use signals from CH F
TRIG_CHAN_G	0x8	Use signals from CH G
TRIG_CHAN_H	0x9	Use signals from CH H
TRIG_CHAN_I	0xA	Use signals from CH I
TRIG_CHAN_J	0xB	Use signals from CH J
TRIG_CHAN_K	0xC	Use signals from CH K
TRIG_CHAN_L	0xD	Use signals from CH L
TRIG_CHAN_M	0xE	Use signals from CH M
TRIG_CHAN_N	0xF	Use signals from CH N
TRIG_CHAN_O	0x10	Use signals from CH O
TRIG_CHAN_P	0x11	Use signals from CH P
TRIG_PXI_STAR	0x100	Use the PXI star trigger

SlopeId1
SlopeId2

[in] Select the sign of the rate of change of the trigger signal with time when it crosses the trigger voltage level that is required to generate a trigger event.

This parameter can be one of the following values.

Identifier	Value	Description
TRIGGER_SLOPE_POSITIVE	1	The trigger engine output goes from low to high when sample values from the trigger source rise above a specified level.
TRIGGER_SLOPE_NEGATIVE	2	The trigger engine output goes from low to high when sample values from the trigger source fall below a specified level.

Level1
Level2

[in] Select the voltage level that the trigger source signal for the specified trigger engine must pass through to generate a trigger event. See the remarks below.

Return value

The function returns ApiSuccess (512) if succeeds.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

The trigger level is specified as an unsigned 8-bit code that represents a fraction of the full scale input voltage of the trigger source: 0 represents the negative limit, 128 represents the 0 level, and 255 represents the positive limit.

For example, if the trigger source is CH A, and the CH A input range is ± 800 mV, then 0 represents a –800 mV trigger level, 128 represents a 0 V trigger level, and 255 represents +800 mV trigger level.

In general, the trigger level code is given by:

```
TriggerLevelCode = 128 + 127 * TriggerLevelVolts / InputRangeVolts.
```

Note that AlazarSetExternalTrigger is used to select the trigger input range if the trigger source is an external trigger signal connected to the TRIG IN connector.

See Also

Trigger control
AlazarSetTriggerDelay
AlazarSetExternalTrigger
AlazarSetTriggerTimeOut
AlazarSetTriggerOperationForScanning

4.1.90 AlazarSetTriggerOperationForScanning

Configure the trigger engines of a board to use an external trigger input and, optionally, synchronize the start of an acquisition with the next external trigger event after AlazarStartCapture is called.

Syntax

```
C/C++
RETURN_CODE
AlazarSetTriggerOperationForScanning (
      HANDLE BoardHandle,
      U32 TriggerSlopeId,
      U32 TriggerLevel,
      U32 Options
);
```

Parameters

BoardHandle

[in] Handle to board.

SlopeId

[in] Select the direction of the rate of change of the external trigger signal when it crosses the trigger voltage level that is required to generate a trigger event.

This parameter can be one of the following values.

Identifier	Value	Description
TRIGGER_SLOPE_POSITIVE	1	Generate a trigger event when the external trigger level rises above a specified level.
TRIGGER_SLOPE_NEGATIVE	2	Generate a trigger event when the exernal trigger level falls below a specified level.

Level

[in] Specify a trigger level code representing the trigger level in volts that an external trigger signal connected must pass through to generate a trigger event. See the Remarks section below.

Optionsl

[in] The options parameter may be one of the following flags

Identifier	Meaning
STOS_OPTION_DEFER_START_CAPTURE (0x00000001)	Wait until the next external trigger event after the application calls

	AlazarStartCapture before arming the board to start the acquisition. The external clock input should be valid when the trigger event arrives.

Return value

The function returns ApiSuccess (512) if succeeds.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

AlazarSetTriggerOperationForScanning is intended for scanning applications that supply both external clock and external trigger signals to the digitizer, where the external clock is not suitable to drive the digitizer in between trigger events.

This function configures a board to use trigger operation TRIG_ENGINE_OP_J, and the source of TRIG_ENGINE_J to be TRIG_EXTERNAL. The application must call AlazarSetExternalTrigger to set the full-scale external input range and coupling of the external trigger signal connected to the TRIG IN connector. An application should call **AlazarSetTriggerOperationForScanning** or AlazarSetTriggerOperation, but not both.

The trigger level is specified as an unsigned 8-bit code that represents a fraction of the full scale input voltage of the external trigger range: 0 represents the negative limit, 128 represents the 0 level, and 255 represents the positive limit.

AlazarSetTriggerOperationForScanning in currently only supported on ATS9462 with FPGA 35.0 or later.

See Also

Trigger control
AlazarSetTriggerDelay
AlazarSetTriggerOperation
AlazarSetExternalTrigger
AlazarSetTriggerTimeOut

4.1.91 AlazarSetTriggerTimeOut

Set the time to wait for a trigger event before automatically generating a trigger event.

Syntax

C/C++

```
RETURN_CODE
AlazarSetTriggerTimeOut(
      HANDLE BoardHandle,
      U32 TimeoutTicks
);
```

LabVIEW

Parameters

BoardHandle

[in] Handle to board.

TimeoutTicks

[in] Trigger timeout in 10 µs units, or 0 to wait forever for a trigger event.

Return value

The function returns ApiSuccess (512) if it was able to retrieve the value of the specified parameter.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

To convert the trigger timeout from seconds to trigger timeout ticks, multiply the timeout value in seconds by 1e5.

For example, a trigger timeout of 1 ms is equal to 1e-3 * 1e5 = 100 ticks.

See Also

Trigger control
AlazarSetExternalTrigger
AlazarSetTriggerDelay
AlazarSetTriggerOperation

4.1.92 AlazarSleepDevice

Control power to ADC devices.

Syntax

```
C/C++
RETURN_CODE
AlazarSleepDevice(
      HANDLE BoardHandle,
      U32 SleepState
);
```

Parameters

BoardHandle

[in] Handle to board.

SleepState

[in] Specify power state of ADC converters.

This parameter can be one of the following values.

Identifier	Value	Description
POWER_OFF	0	Turn off power to ADC devices.
POWER_ON	1	Turn on power to ADC devices

Return value

The function returns ApiSuccess (512) if it was able to retrieve the value of the specified parameter.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

The API automatically powers up all devices when it loads.

See Also

4.1.93 AlazarStartAutoDMA

This function is deprecated. Do not use in new designs.

This routine is used to enable the AUTODMA functionalities of the device. It must be called prior to calling AlazarGetNextBuffer(...).

Syntax

```
C/C++
RETURN_CODE
AlazarStartAutoDMA(
      HANDLE h,
      void* Buffer1,
      U32 UseHeader,
      U32 ChannelSelect,
      long TransferOffset,
      U32 TransferLength,
      U32 RecordsPerBuffer,
      U32 RecordCount,
      AUTODMA_STATUS* error,
      U32 cFlags,
      U32 r2,
      U32 *r3,
      U32 *r4
);
```

Parameters

h

[in] Handle to the device.

Buffer1

[out] Data buffer for the first set of transferred records. Buffer1 should be large enough to contain (RecordsPerBuffer*TransferLength) many 16-bit values (VB-Integer, C&C++-short).

If the Record header is selected (UseHeader = 1) then Buffer1 should be large enough to hold (RecordsPerBuffer*(TransferLength+sizeof(ALAZAR_HEADER)) many 16bit values.

UseHeader

[in] If equal to 1 then the AUTODMA record header will precede each record in the Buffer

ChannelSelect

[in] This parameter may be one of the following identifiers or values.

Identifier	Value	Meaning
CHANNEL_A	1	Single channel mode
CHANNEL_B	2	Single channel mode

CHANNEL_A | CHANNEL_B	3	Dual channel mode

TransferOffset

[in] Transfer offset relative to the Trigger point for each record.

TransferLength

[in] The amount to transfer for each record.

RecordsPerBuffer

[in] The number of records that will be transferred into Buffer1. (Please note the size information in Buffer1 description).

RecordCount

[in] The number of records to be captured during this acquisition. Infinite Record Count can be used to create an endless capture for any AutoDMA mode. To use Inifinite records, set the RecordCount parameter of AlazarStartAutoDMA(…) to 0x7FFFFFFF. It is the user's responsibility to set the criteria for stopping an acquisition. Note that AlazarStartAutoDMA routine will overwrite any previous settings for this parameter with the value passed in the RecordCount parameter (Please note the size information in Buffer1 description).

Error

[out] Error return code.

This error code may be one of the following values.

Identifier	Value	Meaning
ADMA_Completed	0	No errors occurred
ADMA_Success	0	No errors occurred
ADMA_Buffer1Invalid	1	Buffer1 is not a suitable buffer
ADMA_Buffer2Invalid	2	Buffer2 is not a suitable buffer
ADMA_BoardHandleInvalid	3	Board handle is not valid
ADMA_InternalBuffer1Invalid	4	The routine cannot allocate enough memory because system resources are low
ADMA_InternalBuffer2Invalid	5	The routine cannot allocate enough memory because system resources are low
ADMA_OverFlow	6	A hardware overflow occurred
ADMA_InvalidChannel	7	The channel selected is invalid
ADMA_DMAInProgress	8	A memory transfer is in progress
ADMA_UseHeaderNotSet	9	UseHeader must be set
ADMA_HeaderNotValid	10	An invalid header was encountered
ADMA_InvalidRecsPerBuffer	11	RecordCount must be a perfect multiple of RecsPerBuffer

cFlags

[in] Control Flags,{0 = The routine will automatically start the acquisition, 1 = The user application must call AlazarStartCapture to start the acquisition}. The constants available are as follows:

Identifier	Meaning
ADMA_EXTERNAL_STARTCAPTURE 0x00000001	The User must call AlazarStartCapture to start the acquisition
ADMA_TRADITIONAL_MODE 0x00000000	Traditional Auto Dma mode captures
ADMA_CONTINUOUS_MODE 0x00000100	Continuous Streaming mode without trigger
ADMA_NPT 0x00000200	No-Pre-Trigger Auto Dma mode

r2

[in] RESERVED.

r3

[out] RESERVED.

r4

[out] RESERVED.

Return value

See Table 1 for a list of error codes.

Remarks

The user must ensure that Buffer1 is a valid buffer of the appropriate size.

Buffer1 should be large enough to contain (RecordsPerBuffer*TransferLength) many 16-bit values (VB-Integer, C&C++-short).
If the Record header is selected (UseHeader = 1) then Buffer1 should be large enough to hold (RecordsPerBuffer*(TransferLength+sizeof(ALAZAR_HEADER)) many 16bit values.

See Also

AlazarAbortAutoDma
AlazarGetNextAutoDMABuffer
Using synchronous AutoDMA

4.1.94 AlazarStartCapture

Arm a board to start an acquisition.

Syntax

C/C++

```
RETURN_CODE
AlazarStartCapture(
      HANDLE BoardHandle,
);
```

LabVIEW

Parameters

BoardHandle
[in] Handle to board.

Return value

The function returns ApiSuccess (512) if the function completes without error.

The function returns ApiError (603) if the requested number of records in a single-port acquisition exceeds the maximum supported by the digitizer. See remarks below.

Otherwise, the function returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

Only call **AlazarStartCapture** on the master board in a master slave board system.

PCIe digitizers support up to 1000 records per acquisition in single-port acquisition mode. Use dual-port AutoDMA acquisition mode to capture more than 1000 records per acquisition. Dual-port AutoDMA acquisitions support an unlimited number of records per acquisition.

See Also

AlazarBeforeAsyncRead
AlazarStartAutoDMA

4.1.95 AlazarStopAutoDMA

This function is deprecated. Do not use in new designs.

This API is used to inhibit the software from issuing any new DMA request to the device. It is meant as a helper function for the AlazarFlushAutoDMA API function.

Syntax

```
C/C++
Void AlazarStopAutoDMA(HANDLE h);
```

Parameters

h

[in] Handle to board.

Return value

None

Remarks

This function is useful in situations where the application software has multiple threads. The software can call this routine to stop the device from issuing DMA requests in preparation for calling API AlazarFlushAutoDMA.

See Also

AlazarFlushAutoDMA
Using synchronous AutoDMA

4.1.96 AlazarTriggered

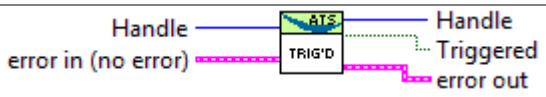
Determine if a board has triggered during the current acquisition.

Syntax

C/C++

```
U32 AlazarTriggered (
      HANDLE BoardHandle
);
```

LabVIEW



Parameters

BoardHandle
[in] Handle to board.

Return value

If the board has received at least one trigger event since the last call to AlazarStartCapture, this function returns 1.

Otherwise, this function returns 0.

Remarks

See Also

AlazarStartCapture

4.1.97 AlazarWaitAsyncBufferComplete

This function returns when a board has received sufficient triggers to fill the specified buffer, or the timeout interval elapses.

Syntax

```
C/C++
RETURN_CODE
AlazarWaitAsyncBufferComplete(
   HANDLE  BoardHandle,
   void *Buffer,
   U32 Timeout_ms
   );
```

Parameters

BoardHandle

[in] Handle to board.

Buffer

[out] Pointer to a buffer to receive sample data from the digitizer board.

Timeout_ms

[in] Specify the time to wait, in milliseconds, for the buffer to be filled.

Return values

If the board receives sufficient trigger events to fill this buffer before the timeout interval elapses, the function returns ApiSuccess (512).

If the timeout interval elapses before the board receives sufficient trigger events to fill the buffer, the function returns ApiWaitTimeout (579).

If the board overflows its on-board memory, the function returns ApiBufferOverflow (582). The board may overflow its on-board memory because the rate at which it is acquiring data is faster than the rate at which the data is being transferred from on-board memory to host memory across the host bus interface (PCI or PCIe). If this is the case, try reducing the sample rate, number of enabled channels, or amount of time spent processing each buffer.

If this buffer was not found in the listof buffers available to be filled by the board, the function returns ApiBufferNotReady (573).

If this buffer is not the buffer at the head of the list of buffers to be filled by the board, this returns ApiDmaInProgress (518).

If the function fails for some other reason, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

You must call AlazarBeforeAsyncRead and AlazarPostAsyncBuffer before calling **AlazarWaitAsyncBufferComplete**.

You must call AlazarAbortAsyncRead before your application exits if you have called AlazarPostAsyncBuffer, and buffers are pending when you wish to exit your application.

Each call to AlazarPostAsyncBuffer adds a buffer to end of a list of buffers to be filled by the board. **AlazarWaitAsyncBufferComplete** expects to wait on the buffer at the head of the list of buffers available to be filled by the board. As a result, you must wait for buffers in the same order that they were posted.

When **AlazarWaitAsyncBufferComplete** returns ApiSuccess (512), the buffer is removed from the list of buffers to be filled by the board.

The arrangement of sample data in each buffer depends on the AutoDMA mode specified in the call to AlazarBeforeAsyncRead.

See Also

AlazarAbortAsyncRead
AlazarBeforeAsyncRead
AlazarPostAsyncBuffer
Using asynchronous AutoDMA

4.1.98 AlazarWaitForBufferReady

This function is deprecated. Do not use in new designs.

This function will stall the current thread of execution for tms number milliseconds or until a buffer has been successfully transferred to a user space AutoDMA buffer. The function must be called after API AlazarEvents(h,1) and before API AlazarGetNextAutoDMABuffer(…). It will wait on the driver to signal the Driver's Internal registered event for up to tms number of milliseconds. When the DMA completes, the signaling event will wake up the Api.

Syntax

C/C++

```
RETURN_CODE
AlazarWaitForBufferReady(
      HANDLE h,
      U32 tms
);
```

Parameters

h

[in] Handle to the device.

tms

[in] time in milliseconds

Return values

670 - signifies that a NULL was used for the handle

671 - signifies that the current device driver does not support events.

672 – Events were not activated using API AlazarEvents.

ApiSuccessFul or 512 signifies that the internal wait event was successfully registered and signaled by the ISR.

ApiFailed or 513 signifies that the internal wait event did not register.

ApiWaitTimeOut or 579 signifies that the internal wait event was not signaled by the ISR.

Remarks

This functionality is only present on the ATS460, ATS660 and ATSS860 devices.
If AlazarEvents(h,1) was not used, calling AlazarWaitForBuffer(...) will return ApiFailed and will not disrupt any ongoing signal captures.

Below is a pseudo-code fragment that shows the operations of API AlazarEvents(…) and API AlazarWaitForBufferReady(…).

Pseudo-code:

```
AlazarSetRecordSize(...);
AlazarSetCaptureClock(...);
AlazarInputControl(...);
AlazarInputControl(...);
AlazarSetTriggerOperation(...)
//
AlazarEvents(h,1);
//
AlazarStartAutoDMA(...);
while (looping == 1)
{
      AlazarWaitForBufferReady(h, 10);
      status = AlazarGetNextAutoDMABuffer();
      if (status == 513)
      {
            looping = 0;
      }
      //Valid data exists in either UserData[0] or UserData[1]
      if ((WhichOne == 0)||(WhichOne == 1))
      {
            //Process Your Data here
            ...
      }
      if (error == ADMA_OverFlow)
      {
            looping = 0;
            returnValue = -4;
      }

}
AlazarCloseAUTODma(...);
//
AlazarEvents(h,0);
//
```

See Also

AlazarEvents
Using synchronous AutoDMA

4.1.99 AlazarWaitNextAsyncBufferComplete

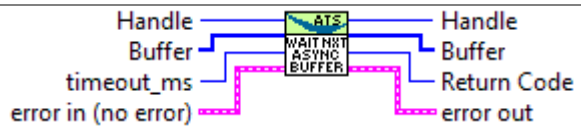
This function returns when the board has received sufficient trigger events to fill the buffer, or the timeout interval has elapsed. To use this function, AlazarBeforeAsyncRead must be called with the ADMA_ALLOC_BUFFERS flag.

Syntax

C/C++

```
RETURN_CODE
AlazarWaitNextAsyncBufferComplete(
   HANDLE  BoardHandle,
   void *Buffer,
   U32 BytesToCopy,
   U32 Timeout_ms
);
```

LabVIEW



Parameters

BoardHandle

[in] Handle to board.

Buffer

[out] Pointer to a buffer to receive sample data from the digitizer board.

BytesToCopy

[in] The number of bytes to copy into the buffer.

Timeout_ms

[in] Specify the time to wait, in milliseconds, for the buffer to be filled.

Return values

If the board receives sufficient trigger events to fill the next available buffer before the timeout interval elapses, and the buffer is not the last buffer in the acquisition, the function returns ApiSuccess (512).

If the board receives sufficient trigger events to fill the next available buffer before the timeout interval elapses, and the buffer is the last buffer in the acquisition, the function returns ApiTransferComplete (589).

If the timeout interval elapses before the board receives sufficient trigger events to fill the next available buffer, the function returns ApiWaitTimeout (579).

If the board overflows its on-board memory, the function returns ApiBufferOverflow (582). The board may overflow its on-board memory because the rate at which it is acquiring data is faster than the rate at which the data is being transferred from on-board memory to host memory across the host bus interface (PCI or PCIe). If this is the case, try reducing the sample rate, number of enabled channels, or amount of time spent processing each buffer.

If the function fails for some other reason, it returns an error code that indicates the reason that it failed. See Table 1 for a list of error codes.

Remarks

You must call AlazarBeforeAsyncRead with the ADMA_GET_PROCESSED_DATA flag before calling **AlazarWaitNextAsyncBufferComplete**.

To discard buffers, set the BytesToCopy parameter to zero. This will cause **AlazarWaitNextAsyncBufferComplete** to wait for a buffer to complete, but not copy any data into the application buffer.

To enable disk streaming using high-performance disk I/O functions, call AlazarCreateStreamFile before calling AlazarWaitNextAsyncBufferComplete. For best performance, set the BytesToCopy parameter to zero so that data is streamed to disk without making any intermediate copies in memory.

If AlazarBeforeAsyncRead is called with the ADMA_GET_PROCESSED_DATA flag, **AlazarWaitNextAsyncBuferComplete** will process buffers so that the data always appears in NPT format: R1A, R2A, … RnA, R1B, R2B, … RnB. This may simply you application, but it comes at the expense of added processing time for each buffer.

If AlazarBeforeAsyncRead is not called with the called with the ADMA_GET_PROCESSED_DATA flag set, then arrangement of sample data in a buffer depends on the AutoDMA mode.

See Also

AlazarAbortAsyncRead
AlazarBeforeAsyncRead
AlazarPostAsyncBuffer
Using asynchronous AutoDMA

4.2 Error Codes

The following table lists the error codes that are returned by the API, their numerical values, and their descriptions. These error codes are declared in AlazarError.h.

Table 1 Error Codes

Identifier	Value	Comment
ApiSuccess	512	The operation completed without error.
ApiFailed	513	The operation failed.
ApiAccessDenied	514	
ApiDmaChannelUnavailable	515	
ApiDmaChannelInvalid	516	
ApiDmaChannelTypeError	517	
ApiDmaInProgress	518	
ApiDmaDone	519	
ApiDmaPaused	520	
ApiDmaNotPaused	521	
ApiDmaCommandInvalid	522	
ApiDmaManReady	523	
ApiDmaManNotReady	524	
ApiDmaInvalidChannelPriority	525	
ApiDmaManCorrupted	526	
ApiDmaInvalidElementIndex	527	
ApiDmaNoMoreElements	528	
ApiDmaSglInvalid	529	
ApiDmaSglQueueFull	530	
ApiNullParam	531	
ApiInvalidBusIndex	532	
ApiUnsupportedFunction	533	
ApiInvalidPciSpace	534	
ApiInvalidIopSpace	535	
ApiInvalidSize	536	
ApiInvalidAddress	537	
ApiInvalidAccessType	538	
ApiInvalidIndex	539	
ApiMuNotReady	540	
ApiMuFifoEmpty	541	
ApiMuFifoFull	542	
ApiInvalidRegister	543	
ApiDoorbellClearFailed	544	
ApiInvalidUserPin	545	
ApiInvalidUserState	546	
ApiEepromNotPresent	547	

Identifier	Value	Comment
ApiEepromTypeNotSupported	548	
ApiEepromBlank	549	
ApiConfigAccessFailed	550	
ApiInvalidDeviceInfo	551	
ApiNoActiveDriver	552	
ApiInsufficientResources	553	
ApiObjectAlreadyAllocated	554	
ApiAlreadyInitialized	555	
ApiNotInitialized	556	
ApiBadConfigRegEndianMode	557	
ApiInvalidPowerState	558	
ApiPowerDown	559	
ApiFlybyNotSupported	560	
ApiNotSupportThisChannel	561	
ApiNoAction	562	
ApiHSNotSupported	563	
ApiVPDNotSupported	564	
ApiVpdNotEnabled	565	
ApiNoMoreCap	566	
ApiInvalidOffset	567	
ApiBadPinDirection	568	
ApiPciTimeout	569	
ApiDmaChannelClosed	570	
ApiDmaChannelError	571	
ApiInvalidHandle	572	
ApiBufferNotReady	573	
ApiInvalidData	574	
ApiDoNothing	575	
ApiDmaSglBuildFailed	576	
ApiPMNotSupported	577	
ApiInvalidDriverVersion	578	
ApiWaitTimeout	579	The operation did not finish during the timeout interval. Try the operation again, or abort the acquisition.
ApiWaitCanceled	580	
ApiBufferTooSmall	581	
ApiBufferOverflow	582	The board overflowed its on-board memory. Try reducing the sample rate, reducing the number of enabled channels, increasing the size of each DMA buffer, or increasing the number of

Identifier	Value	Comment
		DMA buffers.
ApiInvalidBuffer	583	
ApiInvalidRecordsPerBuffer	584	
ApiDmaPending	585	An asynchronous I/O operation was successfully started on the board. It will be completed when sufficient trigger events are supplied to the board to fill the buffer.
ApiLockAndProbePagesFailed	586	The driver or operating system was unable to prepare the specified buffer for a DMA transfer. Try reducing the buffer size, or total number of buffers.
ApiWaitAbandoned	587	
ApiWaitFailed	588	
ApiTransferComplete	589	This buffer is the last in the current acquisition.
ApiPllNotLocked	590	A hardware error has occurred. Contact AlazarTech.
ApiNotSupportedInDualChannelMode	591	The requested number of samples per channel is too large to fit in on-board memory. Try reducing the number of samples per channel, or switching to single channel mode.
ApiNotSupportedInQuadChannelMode	592	The requested number of samples per channel is too large to fit in on-board memory. Try reducing the num
ApiFileIoError	593	A file read or write error occurred.
ApiInvalidClockFrequency	594	The requested ADC clock frequency is not supported.
ApiError	603	The requested number of records in a single-port acquisition exceeds the maximum supported by the digitizer. Use dual-ported AutoDMA to acquire more records per acquisition.
ApiError	604	The requested number of records in a single-port acquisition exceeds the maximum supported by the digitizer.

4.3 Glossary

AutoDMA	AutoDMA is a hardware feature of AlazarTech digitizers. It allows the digitizer to store sample data to its on-board memory at the same time that it performs a DMA transfer from its on-board memory to host memory.
Continuous AutoDMA	Continuous is an AutoDMA mode that configures a digitizer to capture a continuous stream of sample data from one or more analog input channels. The digitizer splits the continuous stream of samples into segments, and transfers segments one at a time to host memory. There can be an infinte number of segments in one acquisition. Trigger engines are disabled in continuous mode AutoDMA. The acquisition begins on a software command.
Dual-port memory	Dual-port memory on AlazarTech digitizers is RAM memory that allows simultaneous read and write operations.
Record	A record is a sequence of ADC values captured as a result of a trigger event. Records may contain sample values that occur both before and after a trigger event.
NPT AutoDMA	NPT is an AutoDMA mode that configures a digitizer to capture multiple records (one per trigger event) with no samples before the trigger event. It supports higher trigger rates than traditional AutoDMA mode.
Traditional AutoDMA	Traditional is an AutoDMA mode that configures a digitizer to acquire multiple records (one per trigger event) with samples before and after a trigger event, and optional record headers including timestamps. It supports lower trigger repeat rates than NPT mode.
Triggered streaming AutoDMA	Trigger streaming is an AutoDMA mode that configures a digitizer to capture a continuous stream of sample data from one or more analog input channels. The digitizer splits the continuous stream of samples into segments, and transfers segments one at a time to host memory. There can be an infinte number of segments in one acquisition. Ttrigger engines are enabled in triggered streaming mode. The acquisition begins when the digitizer receives a trigger event.

5 Appendix – Board Specific Information

Table 2- Samples per Timestamp

	Active channels				
Board	**1 ch.**	**2 ch.**	**4 ch.**	**8 ch.**	**16 ch.**
ATS310	2	1			
ATS330	2	1			
ATS460	2	1			
ATS660	2	1			
ATS850	2	1			
ATS860	4	2			
ATS9350	8	4			
ATS9351	8	4			
ATS9360	16	8			
ATS9373	16	8			
ATS9416	16	8	4	2	1
ATS9440	16	8	4		
ATS9462	2	1			
ATS9625	2	1			
ATS9626	2	1			
ATS9870	16	8			
AXI9870	16	8			

Table 3 - Trigger Delay Alignment

	Active channels				
Board	**1 ch.**	**2 ch.**	**4 ch.**	**8 ch.**	**16 ch.**
ATS310	2	1			
ATS330	2	1			
ATS460	2	1			
ATS660	2	1			
ATS850	2	1			
ATS860	4	2			
ATS9350	8	4			
ATS9351	8	4			
ATS9360	16	8			
ATS9373	16	8			
ATS9416	16	8	4	2	1
ATS9440	16	8	4		
ATS9462	2	1			
ATS9625	2	1			
ATS9626	2	1			
ATS9870	16	8			
AXI9870	16	8			

Table 4 - Input Channel Configurations

	Channels per board		
Channels	2	4	16
A	✓	✓	✓
B	✓	✓	✓
A + B	✓	✓	✓
C		✓	✓
A + C		✓	
B + C		✓	
D		✓	✓
A + D		✓	
B + D		✓	
C + D		✓	
A + B + C + D		✓	✓
E			✓
F			✓
G			✓
H			✓
A + B + C + D + E + F + G + H			✓
I			✓
J			✓
K			✓
L			✓
M			✓
O			✓
P			✓
A + B + C + D + E + F + G + H + I + J + K + L + M + N + O + P			✓

- **2 channel boards:** ATS310, ATS330, ATS460, ATS660, ATS850, ATS860, ATS9350, ATS9351, ATS9360, ATS9373, ATS9462, ATS9625, ATS9626, ATS9870 and AXI9870
- **4 channel board:** ATS9440
- **16 channel board:** ATS9416

Table 5 - Available Impedances and Input Ranges

Identifier	Value	ATS310, ATS330	ATS460, ATS860	ATS660, ATS9462	ATS850
INPUT_RANGE_PM_20_MV	1	-	1MΩ/50Ω	-	1MΩ/50Ω
INPUT_RANGE_PM_40_MV	2	1MΩ/50Ω	1MΩ/50Ω	-	1MΩ/50Ω
INPUT_RANGE_PM_50_MV	3	1MΩ/50Ω	1MΩ/50Ω	-	1MΩ/50Ω
INPUT_RANGE_PM_80_MV	4	1MΩ/50Ω	1MΩ/50Ω	-	1MΩ/50Ω
INPUT_RANGE_PM_100_MV	5	1MΩ/50Ω	1MΩ/50Ω	-	1MΩ/50Ω
INPUT_RANGE_PM_200_MV	6	1MΩ/50Ω	1MΩ/50Ω	1MΩ/50Ω	1MΩ/50Ω
INPUT_RANGE_PM_400_MV	7	1MΩ/50Ω	1MΩ/50Ω	1MΩ/50Ω	1MΩ/50Ω
INPUT_RANGE_PM_500_MV	8	1MΩ/50Ω	1MΩ/50Ω	-	1MΩ/50Ω
INPUT_RANGE_PM_800_MV	9	1MΩ/50Ω	1MΩ/50Ω	1MΩ/50Ω	1MΩ/50Ω
INPUT_RANGE_PM_1_V	10	1MΩ/50Ω	1MΩ/50Ω	-	1MΩ/50Ω
INPUT_RANGE_PM_2_V	11	1MΩ/50Ω	1MΩ/50Ω	1MΩ/50Ω	1MΩ/50Ω
INPUT_RANGE_PM_4_V	12	1MΩ/50Ω	1MΩ/50Ω	1MΩ/50Ω	1MΩ/50Ω
INPUT_RANGE_PM_5_V	13	1MΩ/50Ω	1MΩ/50Ω	-	1MΩ/50Ω
INPUT_RANGE_PM_8_V	14	1MΩ/50Ω	1MΩ/50Ω	1MΩ/50Ω	1MΩ/50Ω
INPUT_RANGE_PM_10_V	15	1MΩ/50Ω	1MΩ/50Ω	-	1MΩ/50Ω
INPUT_RANGE_PM_20_V	16	1MΩ/50Ω	-	-	1MΩ/50Ω
INPUT_RANGE_PM_16_V	18	-	-	1MΩ/50Ω	-
INPUT_RANGE_1_V_25	33	-	-	-	-

Identifier	Value	ATS9325, ATS9350, ATS9850, ATS9870, AXI9870	ATS9351, ATS9360, ATS9373, ATS9416	ATS9625, ATS9626	ATS9440
INPUT_RANGE_PM_20_MV	1	-	-	-	-
INPUT_RANGE_PM_40_MV	2	50Ω	-	-	-
INPUT_RANGE_PM_50_MV	3	-	-	-	-
INPUT_RANGE_PM_80_MV	4	-	-	-	-
INPUT_RANGE_PM_100_MV	5	50Ω	-	-	50Ω
INPUT_RANGE_PM_200_MV	6	50Ω	-	-	50Ω
INPUT_RANGE_PM_400_MV	7	50Ω	50Ω	-	50Ω
INPUT_RANGE_PM_500_MV	8	-	-	-	-
INPUT_RANGE_PM_800_MV	9	-	-	-	-
INPUT_RANGE_PM_1_V	10	50Ω	-	-	50Ω
INPUT_RANGE_PM_2_V	11	50Ω	-	-	50Ω
INPUT_RANGE_PM_4_V	12	50Ω	-	-	50Ω
INPUT_RANGE_PM_5_V	13	-	-	-	-
INPUT_RANGE_PM_8_V	14	-	-	-	-
INPUT_RANGE_PM_10_V	15	-	-	-	-
INPUT_RANGE_PM_20_V	16	-	-	-	-
INPUT_RANGE_PM_16_V	18	-	-	-	-
INPUT_RANGE_1_V_25	33	-	-	50Ω	-

Table 6 - Sample Rates

	ATS310	ATS330	ATS460	ATS660	ATS850	ATS860	ATS9350	ATS9351	ATS9360	ATS9373	ATS9416	ATS9440	ATS9462	ATS9625	ATS9626	ATS9870	AXI9870
1kS/s			✓	✓		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
2kS/s			✓	✓		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
5kS/s			✓	✓		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
10kS/s	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
20kS/s	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
50kS/s	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
100kS/s	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
200kS/s	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
500kS/s	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
1MS/s	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
2MS/s	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
5MS/s	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
10MS/s	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
20MS/s	✓		✓	✓			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
25MS/s		✓			✓	✓											
50MS/s		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
100MS/s			✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
125MS/s			✓	✓		✓	✓	✓				✓	✓	✓	✓		
160MS/s													✓				
180MS/s													✓				
200MS/s									✓	✓							
250MS/s						✓	✓	✓						✓	✓	✓	✓
500MS/s							✓	✓	✓	✓						✓	✓
800MS/s									✓	✓							
1000MS/s									✓	✓						✓	✓
1200MS/s									✓	✓							
1500MS/s									✓	✓							
1800MS/s									✓								
2000MS/s										✓							
2400MS/s										✓							
3000MS/s										✓							
3600MS/s										✓							
4000MS/s										✓							
USER_DEF	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 7 - Sample Rate Identifiers

Sample Rate	Identifier	Value
1kS/s	SAMPLE_RATE_1KSPS	0X00000001UL
2kS/s	SAMPLE_RATE_2KSPS	0X00000002UL
5kS/s	SAMPLE_RATE_5KSPS	0X00000004UL
10kS/s	SAMPLE_RATE_10KSPS	0X00000008UL
20kS/s	SAMPLE_RATE_20KSPS	0X0000000AUL
50kS/s	SAMPLE_RATE_50KSPS	0X0000000CUL
100kS/s	SAMPLE_RATE_100KSPS	0X0000000EUL
200kS/s	SAMPLE_RATE_200KSPS	0X00000010UL
500kS/s	SAMPLE_RATE_500KSPS	0X00000012UL
1MS/s	SAMPLE_RATE_1MSPS	0X00000014UL
2MS/s	SAMPLE_RATE_2MSPS	0X00000018UL
5MS/s	SAMPLE_RATE_5MSPS	0X0000001AUL
10MS/s	SAMPLE_RATE_10MSPS	0X0000001CUL
20MS/s	SAMPLE_RATE_20MSPS	0X0000001EUL
25MS/s	SAMPLE_RATE_25MSPS	0X00000021UL
50MS/s	SAMPLE_RATE_50MSPS	0X00000022UL
100MS/s	SAMPLE_RATE_100MSPS	0X00000024UL
125MS/s	SAMPLE_RATE_125MSPS	0x00000025UL
160MS/s	SAMPLE_RATE_160MSPS	0x00000026UL
180MS/s	SAMPLE_RATE_180MSPS	0x00000027UL
200MS/s	SAMPLE_RATE_200MSPS	0X00000028UL
250MS/s	SAMPLE_RATE_250MSPS	0X0000002BUL
500MS/s	SAMPLE_RATE_500MSPS	0X00000030UL
800MS/s	SAMPLE_RATE_800MSPS	0X00000032UL
1000MS/s	SAMPLE_RATE_1000MSPS	0x00000035UL
1200MS/s	SAMPLE_RATE_1200MSPS	0x00000037UL
1500MS/s	SAMPLE_RATE_1500MSPS	0x0000003AUL
1800MS/s	SAMPLE_RATE_1800MSPS	0x0000003DUL
2000MS/s	SAMPLE_RATE_2000MSPS	0x0000003FUL
2400MS/s	SAMPLE_RATE_2400MSPS	0x0000006AUL
3000MS/s	SAMPLE_RATE_3000MSPS	0x00000075UL
3600MS/s	SAMPLE_RATE_3600MSPS	0x0000007BUL
4000MS/s	SAMPLE_RATE_4000MSPS	0x00000080UL
USER_DEF	SAMPLE_RATE_USER_DEF	0x00000040UL

© 2003-2015 Alazar Technologies Inc. Version 7.1.1

Table 8 - Samples per Record Alignment Requirements

Board type	Minimum record size (samples)	Pretrigger alignment (samples)	Buffer alignment (samples)	Buffer alignment in NPT mode (samples)
ATS310	256	4	16	Not supported
ATS330	256	4	16	Not supported
ATS460	128	16	16	32
ATS660	128	16	16	32
ATS850	256	4	4	Not supported
ATS860	256	32	32	64
ATS9350	256	32	32	32
ATS9351	256	32	32	32
ATS9360	256	128	128	128
ATS9373	256	128	128	128
ATS9416	256	128	128	128
ATS9440	256	32	32	32
ATS9462	256	32	32	32
ATS9625	256	32	32	32
ATS9626	256	32	32	32
ATS9870	256	64	64	64
AXI9870	256	64	64	64

Table 9 – Miscellaneous Features Support

Board	Bandwidth Limit	Input Couplings	8-bit packing	12-bit packing	Configure LSB	
ATS310		AC/DC				
ATS330		AC/DC				
ATS460	✓	AC/DC				
ATS660	✓	AC/DC				
ATS850		AC/DC				
ATS860	✓	AC/DC				
ATS9350		AC/DC				
ATS9351		DC				
ATS9360		DC	✓	✓		
ATS9373		DC	✓	✓		
ATS9416		DC				
ATS9440		AC/DC	✓		✓	
ATS9462	✓	AC/DC				
ATS9625		AC				
ATS9626		DC				
ATS9870	✓	AC/DC				
AXI9870		AC/DC				

Table 10- External Trigger Level Support

Board	1 V	2.5 V	5 V	TTL
ATS310		✓		✓
ATS330		✓		✓
ATS460	✓		✓	
ATS660	✓		✓	
ATS850	✓		✓	
ATS860	✓		✓	
ATS9350			✓	✓
ATS9351			✓	✓
ATS9360		✓		✓
ATS9373		✓		✓
ATS9416		✓		✓
ATS9440			✓	✓
ATS9462	✓		✓	
ATS9625				✓
ATS9626				✓
ATS9870			✓	
AXI9870			✓	

Table 11 – Supported clock types

Board	INTERNAL_CLOCK (1)	FAST_EXTERNAL_CLOCK (2)	MEDIUM_EXTERNAL_CLOCK (3)	SLOW_EXTERNAL_CLOCK (4)	EXTERNAL_CLOCK_AC (5)	EXTERNAL_CLOCK_DC (6)	EXTERNAL_CLOCK_10_MHZ_REF (7)
ATS310	✓	✓					
ATS330	✓	✓					
ATS460	✓	✓	✓	✓			
ATS660	✓			✓	✓	✓	✓
ATS850	✓	✓					
ATS860	✓	✓		✓			
ATS9350	✓			✓	✓		✓
ATS9351	✓			✓	✓		✓
ATS9360	✓				✓		✓
ATS9373	✓				✓		✓
ATS9416	✓				✓		✓
ATS9440	✓	✓		✓			✓
ATS9462	✓			✓	✓	✓	✓
ATS9625	✓				✓		✓
ATS9626	✓				✓		✓
ATS9870	✓			✓	✓		✓
AXI9870	✓			✓	✓		✓

Table 12 - Frequency limits for external clock types (values in MHz unless noted otherwise)

	Fast		Medium		Slow		AC		DC	
	low	high	low	high	low	high	low	high	low	high
ATS310	0	20								
ATS330	0	50								
ATS460	80	125	10	80	0	10				
ATS660					0	10	1k	125	1k	125
ATS850										
ATS860	20	250			0	250				
ATS9350					0	20	1	500		
ATS9351					0	20	1	500		
ATS9360							300	1800		
ATS9373							300	2000		
ATS9416							5	100		
ATS9440	1	125			0	20				
ATS9462					0	10	1	180	1	180
ATS9625							50	250		
ATS9626							50	250		
ATS9870					0	60	200	1000		
AXI9870					0	60	200	1000		

Table 13 - Valid frequencies in PLL mode

ATS660	100-130 MHz in 1 MHz steps
ATS9350	500 MHz
ATS9351	500 MHz
ATS9360	300-1800 MHz in 1 MHz steps
ATS9373	300-2000 MHz in 1 MHz steps (500-2000 MHz in DES mode)
ATS9416	5-100 MHz in 1 MHz steps
ATS9440	125 MHz or 100 MHz
ATS9462	150-180 MHz in 1 MHz steps
ATS9625	250 MHz
ATS9626	250 MHz
ATS9870	1 GHz
AXI9870	1 GHz